IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Donoho *et al* **Docket No.**: UNIV0001D5

Serial No: 09/522,341 **Art Unit**: 2145

Filed: 03/09/2000 **Examiner:** Winder, Patrice L.

Title: Advice Provided for Offering Highly Targeted Advice without Compromising Individual Privacy

January 6, 2006

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Declaration of prior invention to overcome cited patent or publication pursuant to 37 CFR 1.131

1. My name is David Salim Hindawi.

2. I have reviewed the following documents cited by the Examiner as a basis for rejecting the herein claimed invention: U.S. Patents to Hunt, *et al.* (5,893,091)and Wagner (6,092,102).

3. The claimed subject matter of my invention is not what is claimed in the above cited references.

4. Hunt has a critical reference date and filing date of April 11, 1997.

Wagner has a critical reference date and filing date of October 24, 1997.

5. The conception of the claimed subject matter of my invention occurred prior to the specified dates of the cited references and was coupled with due diligence from prior to

said reference date to the filing of the above mentioned application. In support of this, I have attached documents, Exhibits A-F below.

Exhibit A.

Document Title: "AdviceNet Overview"

Document Date: November 22, 1996

This document discloses the present invention including an advice provider which broadcasts information over a communications medium to a third party and a reader resident with the advice consumer. .

Exhibit B.

Document Title: "The AdviceNet System"

Document Date: January, 26, 1997

This document discloses a continuing development of the present invention including the advice provider and resident reader.

Exhibit C.

Document Title: "The AdviceNet Project"

Document Date: March 11, 1997

This document discloses a continuing development of the present invention including the advice provider and resident reader.

Exhibit D.

Document Title: "The AdviceNet Relevance Language"

Document Date: September 18, 1997

This document discloses a continuing development of the present invention including the advice provider and resident reader.

Exhibit E.

Document Title: "The AdviceNet System For Automating Technical Support"

Document Date: December 10, 1997

This document discloses a continuing development of the present invention including the advice provider and resident reader.

Exhibit F.

Document Title: "Outline for Meeting with Michael Glenn"

Document Date: June 08, 1998

The submitted portions of this document disclose an outline of a meeting with Michael Glenn from Glenn Patent Group in preparation for filing of the above-cited application and show continuing diligence in reduction to practice of the present invention.

6. I am the one of the inventors that used each of the attached documents shown in Exhibits A-F. Said documents contain a description of the invention. At least one of the documents was created on November 22, 1996, showing a conception date of the invention prior to the effective date of said references.

7. The above-cited application was filed on March 9, 2000 and is a divisional application of 09/272,937, which was filed on March 19, 1999 which claims benefit of 60/098,798, which was filed on September 1, 1998.

8. The above supporting facts show the conception of the invention prior to the effective date of said references coupled with due diligence from prior to said date to the filing of the above-cited application.

9. I herein acknowledge that willful false statements and the like are punishable by fine or imprisonment, or both (18 U.S.C. 1001) and may jeopardize the validity of my application or any patent issuing thereon. All statements made of my own knowledge are true and all statements made on information or belief are believed to be true.

[illegible]

David SalimHindawi,
Declarant

1/30/06

Date

EXHITIT A

AdviceNet Overview

Friday, November 22, 1996

Introduction

Computer reliability is like the weather: everybody talks about it, but nobody seems to do anything about it. The reason, perhaps. is that while we sometimes perceive reliability as a s single issue, problems with reliability come from a great many sources, and have an enormous variety of solutions.

In fact, many individual reliability problems have been solved, and more solutions are found each day. Hardware vendors replace faulty parts; software authors revise their work; and users themselves experiment with their machines to find ways to keep them working.

Therefore, we propose to attack the problem of reliability from a different angle — we will treat it not as a software-and-hardware problem, but as a communication problem. Rather than solving problems ourselves, we will provide a way to deliver solutions to the people who need them.

The State of the Art

We are not the first to tackle this communication problem. Most users communicate their solutions informally; many reach a wider audience through user groups and bulletin boards; some sell their advice in newsletters or magazine columns. Technical support workers are paid in part to distribute such advice; system administrators are paid in part to collect and sift through the advice others have made available.

But these existing pathways of communication have weaknesses we think we can improve upon. Informal communication reaches a limited audience; information is often corrupted as it spreads in this way. User groups, bulletin boards, newsletters, and advice columns require attention even when their advice is not pertinent. Technical support workers are expensive, and have no effective way to broadcast their advice to those who need it. System administrators are even more expensive, and have limited capacity to remember solutions to rare problems.

Our Solution

We propose to create an automated system which uses the internet to distribute advice. Our plan is to build software which establishes a subscription-based, point-to-point network which will deliver signed advice to users in a mixed machine- and human-readable format.

We envision a subscription-based system because we see the relation between an advisor and an end user as an ongoing one. Advisors not only create fresh advice from time to time, but also update and correct their previous advice. We want users to receive these updates with as little effort as possible.

We envision a point-to-point network because there are so many sources of advice that collecting it all would be a monumental task, and because these are so many users that distributing it all would be still more difficult. A point-to-point network distributes this work over many machines on the internet. Further, it gives us a first layer of screening and security: users will (we hope) not subscribe to advice sources which they find irrelevant or untrustworthy. Finally, it allows for privacy: advice can be provided on a part of the internet which is not globally accessible.

We will provide a second layer of security by insisting that advice be signed. The principal defense a user has against bad advice is knowledge of the source: advice from a trusted source is usually good advice. We will build upon that trust by making sure the source is known, and verifying the source cryptographically.

We want at least some portions of the advice to be machine-readable because we want to automatically filter advice for relevancy. We imagine that most advice will concern particular hardware or software configurations, and that most advice will be irrelevant to most users. A significant part of the value we can provide is in locating that portion of the available advice which is relevant. In the ideal situation, the actions advised will also be machine-executable, so that the user will only be called upon to accept or reject the advice.

On the other hand, we realize that neither the conditions under which advice applies nor the advised actions can always be handled exclusively by the computer; even if they could, a user should have the chance to understand the advice before accepting or rejecting it. Therefore it is essential that advice be provided in a human-readable format.

Delivery

We intend to use HTTP as the primary delivery mechanism for advice. Each user will have a list of URIs to be searched for advice; a demon on the user's machine will periodically check those locations for new advice or updates to old advice. The URI will point to a table of available advice, which will contain the URIs and dates for advice files and other advice tables. This nested approach should allow for a relatively small transaction when small changes are made by an advice supplier.

While the user ought to be able to subscribe to an advice site by typing in its URI, we imagine that the subscription will more typically be created by an internet helper program which interprets the URI by adding it to the subscription list. This will make it easy to advertise advice sites on the web or by email. Also, we will want to provide a mechanism by which a program's installer could subscribe the user to the program's official advice site.

We imagine that the delivery demon collects advice and deposits it in a database on the user's machine (perhaps the same demon could be used to make a mirror of advice sites?) In doing so, it should probably make a judgment as to the possible pertinence of the advice — there's no reason for a user's machine to download or remember advice about products the user doesn't own. But advice about things which work now but could fail later should be kept.

Finally, we imagine that users could also get advice from themselves. After putting together a working system, a user could take a snapshot of it — creating an advice database which can locate, if not repair, changes made to the system by by accident. Of course, since users are unlikely to put in the effort to create and keep up to date an entirely reliable advice database, they should take advice from themselves with a grain of salt.

Checking

We imagine a second demon running on the user's machine, which compares the state of the machine to the the advice database. When the conditions attached to a piece of advice are fulfilled, it reports that advice to the user.

The heart of this comparison seems to be a sort of outline-based pattern matching. We imagine that a system can be described as an outline: it has various pieces of hardware and software packages; the packages have folders and files; the devices, folders, and files have various attributes. Advice will come with patterns attached to it; when the system's outline matches the pattern, the advice is relevant.

While the description above covers the intended result, it's probably not the algorithm we want to use. For efficiency we'll want to only generate those parts of the system description which play a part in the pattern matching.

If done in a sufficiently general fashion, we should be able to build other applications on this same core. One which we have imagined is the composition of a personalized newspaper from a set of rules describing a reader's interests.

Reporting

When a piece of advice becomes relevant, we'll prepare a report for the user, containing, or at least summarizing, all of the relevant advice. Our current feeling is that the report will be built from passages of HTML found in the advice, and presented through a web browser. In this way, we get not only a way to include formatted text and pictures, but also links may be embedded which a user can follow for more complete information.

The advice reports may also contain forms, which the user can use to accept machine-executable advice. In some cases, this will require only pressing a button; in others, a more complex form could be presented.

We can create a more flexible reporting mechanism by using a compound-document format such as OpenDoc. This would allow pieces of advice to present richer user interfaces, and may improve communication between our programs and the advice user interfaces.

Taking Action

While the only action necessary in response to some advice is to inform the user, in other cases we imagine that a full response could be available to the user at the click of a button. A web browser acting alone will not make the changes to a user's system which we imagine we'll want, but it is possible that by including a helper program (whose communication with the web browser is secure against off-machine spoofing) we can take more dangerous action, such as downloading and executing a program which will fix a problem.

We expect to include utilities which will, at a minimum, execute the advice users can generate automatically; we may also include utilities for tasks we expect to be advised by third parties, such as upgrading software.

EXHIBIT B

The AdviceNet System

Sunday, January 26, 1997

The Subscription Editor

I'm thinking we should store each advice subscription as a separate document, which would contain at least these parts:

- the site's URL
- the site's title
- a short description of the site
- the site's recommended update frequency
- the user's chosen update frequency

We'll need a small application which displays this information, allows the frequency to be edited, and can trigger an immediate update. It should probably have an "Update All" switch as well.

This would also be the application to use to create a new subscription by hand, and it should be tied in to Internet Config so that it can display advice sites when other programs find them on the net.

Perhaps this application should be able to make some changes to the advice database as well, such as purging advice from a particular site or adding new advice from files on the local file system.

The Subscription Demon

This demon (which may be in the same background application as the information demon) watches for the subscription list to change or for subscriptions to get out of date. When they do, it tells the advice gatherer to update the advice from the subscription's URL.

The Advice Gatherer

This part goes to the network in response to requests from the subscription editor ("Update Now"), the subscription demon (as subscriptions go out of date), or the information demon (as pieces of advice recommend gathering others). It downloads pieces of advice from the net, parses them, and links their guards into the information network.

Ideally the gatherer would have the usual array of internet protocols at its disposal, but I don't see any reason why HTTP, SHTTP, and local file system access wouldn't suffice.

The Fact Checkers

These parts are called upon by advice guards to examine the state of the system. Each will know how to extract some piece of information from the system, and will have mechanisms for reviewing that information. We should have a plug-in system for these pieces, so that people can check for information we haven't yet imagined. But we should also include some standard kinds of information:

- devices
- computer & CPU model
- amount of RAM
- directory and file info
- system components: applications, control panels, fonts, etc.
- version numbers
- gestalt values
- advice subscriptions
- advice stored in the system
- advice which has been triggered

The Information Network

The information network builds complex facts out of the simple facts and relationships described by the fact checkers. Each node of the network will represent some fact about the system, and links between the nodes will carry updates to the information. When the advice gatherer collects a piece of advice, the guard on that advice will be linked into the network, along with any lesser conditions it depends upon. To keep the network efficient, different guards will be able to share nodes in the network — there may be hundreds of pieces of advice which refer to the system version number, but the version number will have only one node in the network.

The Information Demon

This demon scans the information network, watching for conditions which have changed and updating the facts related to them. When it finds that a guard condition is satisfied, it may ask the gatherer for more advice, or launch the display application to display advice to the user. (Before displaying advice, it should probably flush the information network for more advice that has become pertinent.)

The Display Application

The display application is launched by the information demon when a piece of advice becomes pertinent. It will display the advice's message (presumably in HTML, but we ought to support some other MIME types) and the associated signature.

It will also provide an extension — presumably a new form submission method — which can trigger effects not usually available to HTML. We want advice, once accepted, to be able to download and execute an arbitrary native application.

The HTML will usually contain the URLs and checksum of an application to run, and of files to be handed to that application as input. In addition, the application will receive the form result, if any. Through this mechanism and the use of hidden fields the effect application can also get information on how the advice was triggered.

In addition to the options the advice offers to the user, the display application should provide some standard options. When the user sees a piece of advice, he should always be able to postpone it, ignore it, have the system forget it, check for updates to it, or cancel his subscription to the site it came from.

Finally, the display application should be able to display and check the digital signatures on advice.

The Effects

Once accepted, advice acts on the system by launching applications. Any application could be launched as the effect of a piece of advice, but I imagine that some small applications will be written specifically for advice, and thus take advantage of information about the advice which we make available to them. These applications can be written in either a traditional programming language or in any general-purpose scripting language available on the target system.

While a piece of advice can download its effect application from the net, it's best if the applications for executing some common kinds of advice are available locally. The only three kinds of applications I've thought of that we need are downloading files to install, locating misplaced files, and changing advice subscriptions.

Central Advice Databases

There are a few kinds of advice we should probably provide ourselves, either as part of the bootstrapping process or as a service to enhance the usability of AdviceNet:

- Site Announcements, which would contain advice like "If you have MacWrite, you may want to subscribe to the Claris advice site."
- Urgent Updates, containing advice like "If you subscribe to the Claris site, and you downloaded advice from it yesterday, you should check it again immediately, because they told us they published something really stupid."
- The Better Advice Bureau, which gives advice like "If you subscribe to EvilCorp, you should know that we get lots of complaints about the advice they give out. We think you should stop your subscription and purge their advice from your system."

Automated Advice Writers

Some kinds of advice can be so detailed that it's much easier to extract the right advice from a working machine than it is to write it by hand. In particular, a thorough check of a complicated software installation may be difficult to write by hand. Also, some kinds of advice, such as a check of the system and hardware requirements for a piece of software, could be written just by filling out a form. So we should write a few applications which produce advice by examining part of an example machine, or by questioning the advice author. The output can then be tweaked into the advice the user needs.

Direct Advice Installation

We'd like to provide a mechanism for a software installer to directly add advice to the database. That way the big lump of advice that comes with a new software package won't require a lengthy download.

EXHIBIT C

The AdviceNet Project

Tuesday, March 11, 1997

Introduction

Computer reliability is like the weather: everybody talks about it, but nobody seems to do anything about it. The reason, perhaps. is that while we sometimes perceive reliability as a s single issue, problems with reliability come from a great many sources, and have an enormous variety of solutions.

In fact, many individual reliability problems have been solved, and more solutions are found each day. Hardware vendors replace faulty parts; software authors revise their work; and users themselves experiment with their machines to find ways to keep them working.

Therefore, we propose to attack the problem of reliability from a different angle — we will treat it not as a software-and-hardware problem, but as a communication problem. Rather than solving problems ourselves, we will provide a way to deliver solutions to the people who need them.

The State of the Art

We are not the first to tackle this communication problem. Most users communicate their solutions informally; many reach a wider audience through user groups and bulletin boards; some sell their advice in newsletters or magazine columns. Technical support workers are paid in part to distribute such advice; system administrators are paid in part to collect and sift through the advice others have made available.

But these existing pathways of communication have weaknesses we think we can improve upon. Informal communication reaches a limited audience; information is often corrupted as it spreads in this way. User groups, bulletin boards, newsletters, and advice columns require attention even when their advice is not pertinent. Technical support workers are expensive, and have no effective way to broadcast their advice to those who need it. System administrators are even more expensive, and have limited capacity to remember solutions to rare problems.

Our Solution

We propose to create an automated system which uses the internet to distribute advice. Our plan is to build software which establishes a subscription-based, point-to-point network which will deliver signed advice to users in a mixed machine- and human-readable format.

We envision a subscription-based system because we see the relation between an advisor and an end user as an ongoing one. Advisors not only create fresh advice from time to time, but also update and correct their previous advice. We want users to receive these updates with as little effort as possible.

We envision a point-to-point network because there are so many sources of advice that collecting it all would be a monumental task, and because these are so many users that distributing it all would be still more difficult. A point-to-point network distributes this work over many machines on the internet. Further, it gives us a first layer of screening and security: users will (we hope) not subscribe to advice sources which they find irrelevant or untrustworthy. Finally, it allows for privacy: advice can be provided on a part of the internet which is not globally accessible.

We will provide a second layer of security by insisting that advice be signed. The principal defense a user has against bad advice is knowledge of the source: advice from a trusted source is usually good advice. We will build upon that trust by making sure the source is known, and verifying the source cryptographically.

We want at least some portions of the advice to be machine-readable because we want to automatically filter advice for relevancy. We imagine that most advice will concern particular hardware or software configurations, and that most advice will be irrelevant to most users. A significant part of the value we can provide is in locating that portion of the available advice which is relevant. In the ideal situation, the actions advised will also be machine-executable, so that the user will only be called upon to accept or reject the advice.

On the other hand, we realize that neither the conditions under which advice applies nor the advised actions can always be handled exclusively by the computer; even if they could, a user should have the chance to understand the advice before accepting or rejecting it. Therefore it is essential that advice be provided in a human-readable format.

Delivery

We intend to use HTTP as the primary delivery mechanism for advice. Each user will have a list of URIs to be searched for advice; the user's machine will periodically check those locations for new advice or updates to old advice. The URI will point to a table of available advice, which will contain the URIs and dates for advice files and other advice tables. This nested approach should allow for a relatively small transaction when small changes are made by an advice supplier.

While the user ought to be able to subscribe to an advice site by typing in its URI, we imagine that the subscription will more typically be created by an internet helper program which interprets the URI by adding it to the subscription list. This will make it easy to advertise advice sites on the web, by email, or through other advice sites. Also, we will want to provide a mechanism by which a program's installer could subscribe the user to the program's official advice site.

We imagine that the program will collect advice and deposit it in a database on the user's machine. In doing so, it may make a judgment as to the eventual pertinence of the advice — there's no reason for a user's machine to download or remember advice about products the user doesn't own. But advice about things which work now but could fail later should be kept.

Checking

Some advice which is not relevant when it is first loaded into the machine may become relevant with the passage of time. In particular, advice which describes a properly-installed piece of software may be triggered when a file is moved or modified. To handle this possibility, the program will need to continually compare the state of the machine to the advice in the database. When the conditions attached to a piece of advice are fulfilled, it will report that advice to the user.

The heart of this comparison seems to be a sort of outline-based pattern matching. We imagine that a system can be described as an outline: it has various pieces of hardware and software packages; the packages have folders and files; the devices, folders, and files have various attributes. Advice will come with patterns attached to it; when the system's outline matches the pattern, the advice is relevant.

While the description above covers the intended result, it's not the algorithm we want to use. For efficiency we'll only generate those parts of the system description which play a part in the pattern matching.

Reporting

When a piece of advice becomes relevant, we'll prepare a report for the user, containing, or at least summarizing, all of the relevant advice. Our current feeling is that the report will be built from passages of MIME content such as text or HTML, and presented in a manner similar to email, giving the user an "inbox" of relevant advice. We'll allow advice authors to categorize advice, which will then be sorted automatically into folders within the inbox.

To simplify the presentation of solutions to problems, we will allow advisories to be adorned with buttons which trigger scripts written in some system-specific scripting language. Advisories written in HTML may contain links to web pages and forms, allowing the user to engage in an online dialog with a web server. We may also want to allow advisories to present a more complicated user interface written in Java.

Taking Action

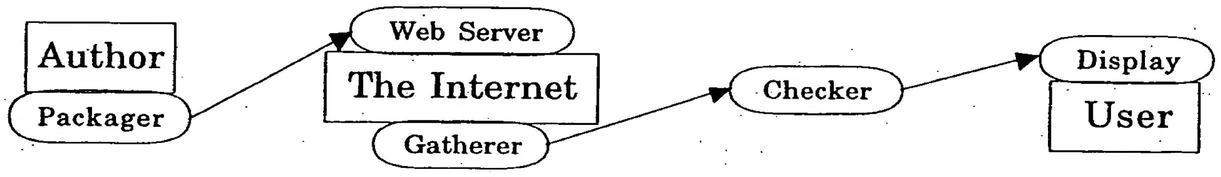
While the only action necessary in response to some advice is to instruct the user, in other cases we imagine that a full response could be available to the user at the click of a button. We want to provide advice authors the ability to execute scripts in the system's native scripting language, and to download and execute programs available on the net.

In addition, we expect to include utilities which will handle common advice actions, and won't have to be downloaded from the net. These actions include locating and updating files, changing advice subscriptions, and purging advice files from the user's database.

Examples

To solidify the way the system works, we'll follow several pieces of advice from the author through their eventual resolution. We'll start with a very basic piece of advice: some printers have been shipped with a defective part, and the user can ask for a replacement.

This advice follows a simple path from the author to the user:



To start the process, the author writes the advice, using a program we'll supply which will have a user interface similar to an email editor. Here's what the author will write:

```
Subject: Some LaserWriter IINT Printers are defective
Guard: exists printer where (model of it is "LaserWriter IINT")
Display-Path: Printer

LaserWriter IINT Printers with serial numbers between 123 and 456
are defective. Call (800)123-4567 for a replacement part.
```

The guard line describes a condition which must be met before this piece of advice will be considered relevant: in this case, there must be a LaserWriter IINT printer available to the system. The display-path line categorizes this advice, putting it into the folder "Printer" folder within the advice inbox.

The advice editor packages this advice by adding lines to the header:

```
From: user@host.com (The author)
Date: Tue, Mar 4, 1997 12:30 PST -0800
Subject: Some LaserWriter IINT Printers are defective
Guard: exists printer where (model of it is "LaserWriter IINT")
Display-Path: Printer
Signed: 89579857946965897893274986526589
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

LaserWriter IINT Printers with serial numbers between 123 and 456
are defective. Call (800)123-4567 for a replacement part.
```

The only interesting line added here is the signature line. We want to encourage advice authors to digitally sign their advice, so that users will have greater confidence in the advice they receive.

Next, the author makes the advice available, by placing it in on a web site to which owners of these printers are expected to subscribe. In this case, an advice site would presumably be maintained by the company selling printers, and users would have either had a subscription installed with their printer software, or have found the advice subscription through the internet.

The user's machine, during a routine check for advice updates, will notice this advisory and download it. In the process, it will add a line indicating the source and time of arrival of the advice:

```
Received: from http://www.apple.com/PrinterAdvice.adv
    by localHost (123.234.123.234);
    Mon, Mar 10, 1997 14:30:16 PST -0800
```

Upon receipt, the advice is checked for relevance; that is, the guard lines on the advisory are tested. If all the guard lines can be evaluated and are satisfied, the advisory will be listed in the display application. If not, it will be filed and rechecked periodically until it expires.

The advisory will wait in the user's advice inbox, filed under its display path "Printer" until either the user opens it or it becomes irrelevant. Eventually, the user will read it, decide how to respond to the advice, and throw it away.

The preceding example is of the simplest kind of advice, a plain text message. Now we'll move on to some more complicated pieces of advice.

Our next example is of a free minor update to an application. This piece of advice has two guard lines, both of which must be satisfied before the advice is considered relevant. It also suggests a response: an updater program is available on the web.

```
Subject: A new version of MicroPhone is available
Guard: version of application id 'DFBO' >= version "4.0"
Guard: version of application id 'DFBO' < version "4.0.3"
Display-Path: MicroPhone
Button: "Install"
Script: tell application "Downloader" to run
     "http://www.svcdudes.com/MicroPhone/updates/MicroPhone 4.0.3"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 59

Version 4.0.3 of MicroPhone is now available.  It fixes several
bugs, including blah, blah, and blah.  You may upgrade to this new
version for free.
```

When the display application shows this advice, it will put a button named "Install" next to the text of the message. If the user clicks this button, the script will run, downloading the updater and running it. The downloader application is one of the simple applications we will provide for executing standard kinds of advice.

Sometimes there's more than one way to respond to an advisory. An advice writer can include more than one MIME section in a single advisory to produce a list of responses. Each section can have its own button and script; when the user chooses a button, the script is executed.

In this example, a file associated to the application has been deleted or moved from its proper location. Three options are presented: search for the file on local disks; download a replacement from the internet; or reinstall the file from the application disks.

```
Subject: MPToolbox is Missing
Guard: version of application id 'DFBO' >= version "3.0"
Guard: !exists file "MPToolbox"
        of parent directory of application id 'DFBO'
Display-Path: MicroPhone
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

The file "MPToolbox" is not in the MicroPhone folder. Without it,
some MicroPhone scripts will not operate correctly.

Button: "Search"
Script: tell application "Searcher" to launch
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

The file may have been moved to another place on the disk.  If so,
the problem can be solved by moving it back to the MicroPhone
folder.

Button: "Download"
Script: tell application "Downloader" to
    copy from "http://www.svcdudes.com/MicroPhone/files/MPToolbox"
    to parent directory of application id 'DFB0'
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

You can solve this problem by downloading "MPToolbox" from the
Software Ventures web site.

Button: "Reinstall"
```

```
Script: tell application "Finder" to
          open alias "MP Install 1:Install MicroPhone"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

This problem can be solved by reinstalling MicroPhone from disks.
```

While the advisory's window is open, the display application will closely watch the guard on the advisory; if the problem is fixed, the advisory will become irrelevant and the window will be closed. If the problem is not fixed, the user can choose another option.

Sometimes there are responses which are only appropriate to some users having a problem. Therefore, we will allow guards to be placed on sections of an advisory. Those sections will be shown only to users whose machines satisfy the guards.

In this example, an application has been modified; the advice detects the modification by using a checksum. The problem can be solved by reinstalling the program, but if the user has the virus checking application Disinfectant, the advisory suggests checking the application for viruses.

```
Subject: MicroPhone has been altered or damaged
Guard: version of application id 'DFBO' = version "4.0.2"
Guard: checksum of application id 'DFBO' != "AB76E48CF09B533F"
Display-Path: MicroPhone
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 59

The application "MicroPhone" has been altered or damaged.

Guard: exists application id 'D2CT'
Button: "Disinfect"
Script: tell application id 'D2CT'
               to open application id 'DFB0'
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 80

This problem may have been caused by a computer virus.  You can
check for known computer viruses by running "Disinfectant."
```

```
Button: "Reinstall"
Script: tell application "Finder" to
          open alias "MP Install 1:Install MicroPhone"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

This problem can be solved by reinstalling MicroPhone from the
original disks.
```

In this case, the first response is unlikely to solve the problem, and is not presented as a solution, but just as a suggestion. As before, the advisory window will stay open until the user dismisses it or the problem is fixed.

Sometimes, a plain text interface is insufficient to convey the content of an advisory, or a simple button is not enough of a user interface to provide a proper response to the advice. For these cases, we will offer HTML as an alternative to plain text. In the example below, an HTML form allows a user to order an upgrade to an application over the web. For those users who don't care to order over the web, a phone number is provided; if the computer can dial the phone, it offers to dial that number.

```
Subject: A new version of MicroPhone is available
Guard: version of application id 'DFBO' < version "4.0"
Display-Path: MicroPhone
MIME-Content-Type: TEXT/HTML
MIME-Content-Length: 276

<html>
Version 4.0.3 of MicroPhone is now available.  This new version
has cool features and is fully buzzword-enabled. You can upgrade
to this new version for a tidy sum.<p>
<form method=post
     url="shttp://www.svcdudes.com/cgi-bin/upgrade.cgi">
Credit card:
<select name="CardType">
     <option>American Express</option>
     <option>Discover</option>
     <option>Mastercard</option>
```

```
    <option>Visa</option>
</select><br>
Number: <input type=text name="CardNumber"><br>
Expiration date: <input type=text name="Expiration"><br>
<input type=submit name="Purchase">
</form></body></html>

Guard: exists extension "Telephony"
Button: "Dial"
Script: tell application "PhoneDialer" to call "(800)734-6727"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

You can also order Microphone by phone at (800)734-6727.

Guard: !exists extension "Telephony"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

You can also order Microphone by phone at (800)734-6727.
```

Our final example doesn't show off any further features of the system, but rather demonstrates that this system can be used in conjunction with a program's existing error-checking mechanisms to get users with problems in touch with technical support. In this example, a piece of advice looks for an error log, and suggests that the user deliver the log to the maintainers of the program:

```
Subject: MicroPhone reports unexpected errors
Display-Path: MicroPhone
Guard: exists file "MicroPhone Error Log"
        of parent directory of application id 'DFBO'
Button: "Send Report"
Script: tell application "Uploader" to submit
     file "MicroPhone Error Log"
        of parent directory of application id 'DFBO'
     to "http://www.svcdudes.com/cgi-bin/bugReporter.cgi"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 59
```

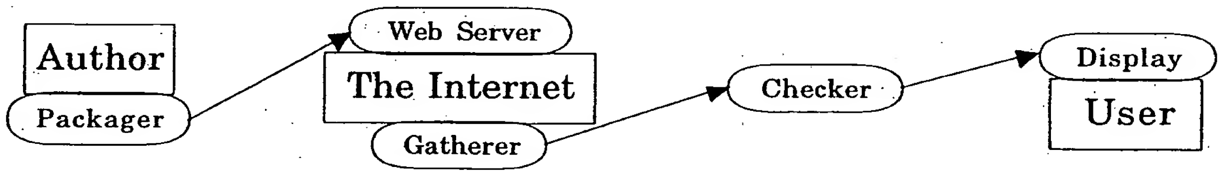
```
MicroPhone has encountered problems which were not fully
anticipated by the programmers. The program has compiled a report
of these problems. To help them fix these problems, please send
this report to Software Ventures.

Guard: size of file "MicroPhone Error Log"
        of parent directory of application id 'DFBO' > 100000
Button: "Delete"
Script: tell application "Finder" to move
     file "MicroPhone Error Log" to the trash folder
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 59

The reports of MicroPhone's problems have grown to more than
100K.  If you do not wish to send these reports to Software
Ventures, you may throw them away.
```

The Central Parts

The basic path advice takes from its author to its recipient passes through five major pieces of software, four of which we'll write:



In the examples above, we've seen how these parts work together to deliver advice to the user. In the descriptions below, we'll see in more detail how the parts themselves will work. The descriptions are arranged in the order I expect they should be written, working away from the user.

The Display Application

The display application is where the user interacts with advice. Its main interface is similar to an email reader, presenting a list of advisories arranged by subject, date of reception, or date of relevancy. The user will have two three-way choices for filtering the advice. The display can be set to show read, unread or both read and unread advisories, and can independently be set to show relevant, irrelevant, or both relevant and irrelevant advisories; the default will be to display all relevant advisories. The advisories will be further sorted by their display paths, producing a built-in hierarchy of folders.

The display application will also be able to open advice files directly, rather than as part of the subscription and system-monitoring mechanism. When the user opens an advice file directly, it will be displayed in a separate window from the inbox, but in other respects it will operate similarly.

When an advisory is opened, its signature and contents will be shown, including any buttons associated to its sections. When the application is set to show irrelevant advice, those sections which are irrelevant will be distinguished visually, and their buttons will be disabled.

While the display application is running, it will check continually for relevant advice becoming irrelevant. As it notices these changes, it will remove the advice from the list (if irrelevant advice is not being shown) and will close any displaying the advisory.

The display application will also allow the user to add or remove advice from the system, check for updates, or to change advice subscriptions. In particular, when viewing an advisory, the user should be able to directly throw it away, check it for updates, or cancel the subscription which brought it.

The display application is probably the most complicated part of the advice system; even apart from the need to include an HTML display engine, it may take three months or more to have all of its basic functionality in place.

The Advice Checker

The advice checker's job is to keep track of which advice is relevant and which advice is irrelevant. It isn't so much an application as a module used by the display application to evaluate guards and (depending on the updating model we choose) by a demon which updates the values of the guards when the display application isn't open.

The difficulty of writing the advice checker is connected to the guard language. I expect we could use AppleScript as a guard language for prototype purposes, and have the checker working in a matter of a few weeks. However, AppleScript probably won't suffice for a shipping version; using it in guards seems to present too many security problems. Putting together a checker with a full language interpreter could take two to four months.

Associated with the checker is a mechanism for extracting basic facts from the system; in some sense, these basic relationships give a vocabulary to go with the advice checker's grammar. We'll want to have a plug-in interface to the basic facts, which will take about two weeks to produce; in addition we'll want to provide some basic facts, including:

- devices
- computer & CPU model
- amount of RAM
- directory and file info
- system components: applications, control panels, fonts, etc.
- version numbers
- gestalt values
- advice subscriptions
- advice stored in the system
- advice which has been triggered

The individual checkers will be simple, with most taking less than a day to produce.

The Advice Gatherer

This is the part which which will download advice from the net. Its heart will be an implementation of HTTP and SHTTP. I imagine that the HTTP can be up and running in about two month's work; while I'm less familiar with SHTTP, I suspect it will take less than one month more.

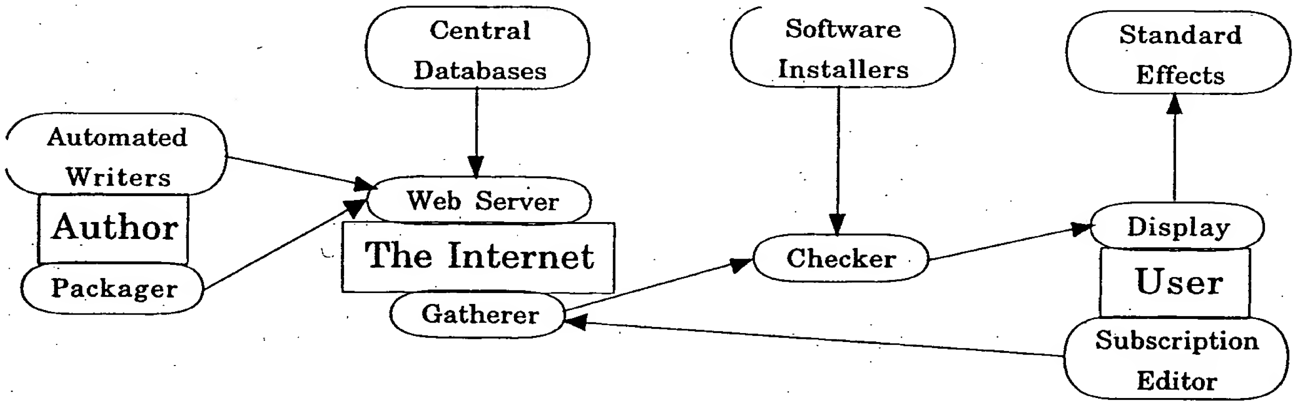
The Advice Packager

The packager is the basic authoring tool; it will work much like an email editor combined with the advice display application. (It may even be appropriate to sell the packager as an enhanced version of the client program, as with Netscape.) The basic interface is a text editor with separate fields for the header lines which the author can fill in. When the advice has been written, the packager fills in the computer-generated header lines.

In its basic form, the packager is a simple application, and should take less than a month to write. If we expand its definition to include editing HTML, it becomes a much more lengthy project.

The Peripheral Parts

While the parts above form the core of the advice system, there are several other parts we want to add to give the system a richer flavor.



Subscription Editor

The subscription editor is the interface through which the user can add and remove subscriptions. For each site the user subscribes to, the subscription editor will keep track of

- the site's URL

EXHIBIT D

The AdviceNet RELEVANCE Language

Universe Communications, Inc.
2180 Dwight Way, Suite C
Berkeley CA 94704

9/18/97

Abstract

The RELEVANCE Language is a key component of the ADVICENET system, enabling authors of advisories to specify with precision the computers for which a certain advisory is relevant. This document describes the formal structure of the language and places in context the many considerations which have led to the current structure.

22 PAGES

Contents

1 Preliminaries

This document describes the RELEVANCE language of ADVICENET . It aims only to provide basic information about the structure and capabilities of the language. It represents version α .10 and was authored on 9/18/97 .

A full appreciation of the language's use requires an understanding of the ADVICENET system, and can be obtained by careful study of the following documents.

- THE ADVICENET SYSTEM. Describes the current crisis in computer reliability and maintenance. Describes the ADVICENET system for communicating active advisories to computers worldwide. Describes the impact of this system on components of the existing crisis. Describes the development effort needed for a computer software/hardware manufacturer or a corporate intranet adminsitrator to benefit from ADVICENET .

- THE ADVICENET SITE DEVELOPER'S MANUAL. Describes the components of an advice site and how an advice provider can construct those components. Illustrates how an advice site can save an advice provider money in technical support and maintenance costs.

- THE ADVICENET INSPECTOR API. Describes how an advice provider can extend the RELEVANCE language to include capabilities directly addressing specific needs of the advice provider. Gives detailed information on the Applications Programming Interface and on the programming environment which is required to develop Inspectors.

- THE ADVICENET MAC OS INSPECTOR LIBRARY. Describes a layer of Macintosh-specific Types, Properties, Elements, and Casts which supplements the base RELEVANCE language with a comprehensive range of services for inquiring about the state of a MacOS computer.

- THE ADVICENET WIN95 INSPECTOR LIBRARY. Describes a layer of Windows '95-specific Types, Properties, Elements, and Casts which supplements the base RELEVANCE language with a comprehensive range of services for inquiring about the state of a Wintel '95 computer.

These documents are currently still under development.

2 Role of the RELEVANCE Language in AdviceNet

To set the stage for a detailed description of the RELEVANCE language, we begin with a sketch of the role played by the language in ADVICENET .

The ADVICENET system is based on the idea of a decentralized community of advice providers offering *advice sites* distributed worldwide throughout the Internet. These advice sites make available for HTTP access special documents called *advisories*. Advisories are ASCII text files resembling very much in format existing e-mail messages; in fact they are custom extensions of the e-mail/MIME message formats offered through Internet Standard RFC 822. It is expected that at some future date, the ADVICENET extensions will have proven so widely useful that they will become Internet standards themselves.

Advisories have a format resembling the format of e-mail messages, with many of the same components in the message/digest headers. The key extension offered by advisories is the institution of a new clause in the message: the *relevance clause*. The relevance clause is distinguished by the keyword `X-Relevant-When:`; what follows the keyword is an *expression* from the RELEVANCE language. It is this language that is being described in this technical report.

The purpose of this language is to enable the ADVICENET *Advice Reader*, running on a user's computer to *automatically* read an advisory and determine, *without intervention from the user*, whether the advisory is *relevant* to the user. Under the mediation of various user interface parameters set in the Advice Reader, the user will typically be notified of relevant advisories, and will be given the opportunity to *follow-through* as the advisories suggest. The actual follow-through action will depend on the advisory, but typically this will be an option to obtain and execute an application which renders, in a fully-automatic way, a repair to an undesirable situation.

This scheme means that it is possible for an advice author to publish advisories at its web site that solve many of the most common technical support problems faced by clients of the author's organization, and be assured that the advisories come to the attention only of users of precisely the computers that are in need of the specific advice being offered. The value in this procedure is that it saves humans the time and trouble of precisely matching solutions to computers in need of those solutions.

With this background, the RELEVANCE language can be seen as fitting into a context of

(i) Widespread distribution via Internet;

(ii) Unattended automatic parsing and evaluation;

(iii) User notification of *relevant* advisories only.

It is particularly important to keep in mind that advice is evaluated for relevance *continually*. That is to say, andvisory, once loaded into a computer's advice reader, can reside in the advice reader for an unlimited amount of time, and can continue to be checked day after day, imposing a de facto constant vigil watching the system state for conditions as they develop. Hence it is particularly important that the system be designed with unattended operation in mind, and that unatteded operation be reliable and even confidence-inspiring.

3 Design Goals of AdviceNet

The RELEVANCE language has been very specifically designed with this application background and purpose in mind. The design as it stands today aims at combining the following characteristics.

- *English-Accessible.* A relevance clause is, in principle, something an individual user could read and comprehend, though few users will choose to do so in most cases. The syntax of the RELEVANCE language resembles the syntax of plain english, with key roles in the language played by clauses formed from articles like *of*, *as*, *whose*, *exists*, and so on.

- *Descriptive.* The purpose of a relevance clause is to examine the state of an individual computer, and see whether it meets various conditions which could combine to imply the relevance of a certain advisory. In short the purpose of the language is to describe rather than to manipulate.

- *Nonprocedural.* Unlike many languages used in connection with the operation and/or maintenance of computers, the RELEVANCE language is not procedural: it does not specify how to manipulate the contents of various fragments of memory. This is the flip side of being descriptive: there is no useful purpose that would be served by enabling traditional procedural services: loops, assignments, and conditionals. Such services are not made available.

- *Open-ended.* The purpose of the language is to precisely describe the state of a computer. This state can change as the user purchases new software and/or hardware, or indeed as new software/hardware objects are invented. Consequently it is not possible to limit in advance what are all the components of state that the language will give access to; and the language is designed to give authors the ability to extend the language to express concepts about system state that haven't yet been conceived of.

- *Object-Oriented.* The language is extensible using the discipline of object oriented programming (OOP). The OOP buzzword means, in our case, that we offer a *Strongly-Typed, Polymorphic* language. We follow the strong-typing philosophy in the extreme. We carefully define the collection of all objects one would like to query, and carefully specify the methods for making queries, and forbid combinations that cannot be formed using our controlled collection of methods. We aim to avoid the usual procedural approach offering powerful, flexible, general purpose tools which can be used in many unanticipated ways. In the usual cases where that approach is used, it is good, but in the setting we are studying, it would be *dangerous.* OOP methodology allows us to design a controlled environment enabling powerful queries of the system state while remaining relatively secure from misuse and abuse.

- *Security and Privacy Aware.* Because the language is designed to be used in conditions of unattended operation over the Internet, it is very important that the language be safe in various ways. This concern has overwhelmed all others in the design of the product; it is fair to say that each of the distinctive features of the RELEVANCE language has been chosen based on these important concerns. It is believed that the language itself best addresses the security and privacy concerns that Internet operation will demand.

4 Security Issues in AdviceNet

The Internet is not a secure medium; along with the many services which it provides come the risks of exposing one's machines to potentially erroneous or even malicious agents. In designing the ADVICENET system, with its aspects of decentralized authorship and unattended operation, it is crucially important for the acceptance and usefulness of the system to address Internet security concerns.

We begin with the obvious comment that there is no such thing as an "ironclad" guarantee of security. The RELEVANCE language has been designed with several security issues in mind; the design addresses these issues, though there remains the possibility of other concerns we have not thought of. Your mileage may vary.

Our main concern is that the evaluation of clauses in the language not have damaging or embarassing side-effects. In evaluating clauses of the language, ideally we would have these four properties:

1. *No significant Claim on system resources.* Evaluation of a relevance clause should not consume inordinate amounts of CPU time, CPU memory, or hard drive space.

2. *No permanent change in system state.* The evaluation of clauses cannot effect the system, for example through creating or destroying files.

3. *No violation of user privacy.* The evaluation of clauses cannot result in an advisory communicating private data about the user to the outside world.

4. *Robustness against misuse of the language.* Desiderata 1-3 are not only supposed to hold in evaluation of well-formed clauses authored by those with "friendly intent". The evaluation of *arbitrary* clauses, even those which are malformed or which are authored by agents with "hostile intent", should *never* be able to lead to damaging side-effects. For example, the evaluation should be safe despite subtle mis-uses of the language – improperly formed clauses, misapplied operators, or mismatching data types.

These different concerns have dictated various aspects of the language design. For example, the RELEVANCE language is nonprocedural; this responds to security concerns, as it places limits on the resources which evaluation can consume. At an elementary level, the RELEVANCE language does not allow for recursion, for infinite loops, for arbitrarily-sized arrays to be allocated; these limits allow explicit bounds to be known (in principle) about the resources which a relevance clause can consume. A famous problem in logic concerns the ability to predict whether a machine evaluating an expression in a procedural language will *ever* reach completion. This is the *Turing Halting Problem.* It is not in general possible to decide whether a program in a procedural language will ever terminate. The situation in our setting stands in stark contrast:

All RELEVANCE *expressions are decidable: they must halt.*

The object-oriented nature of the language also responds to security concerns. As the language is strongly-typed, it is not possible to apply operators to data which they were not designed to support. This allows to avoid crashes and dangerous misreferences, and to safely terminate the evaluation of expressions that are poorly formed.

A second facet of the object-oriented design which may be viewed as a security feature is the fact that manipulations of data are obtained not in the *surface level* of the RELEVANCE language itself but rather in the deeper level of the member functions of data types. These can be expected to be programmed more carefully than would be customary in the programming of a script, and to pay closer attention to clever use of computational resources. Moreover, they provide services which are available only after they are explicitly installed by the user, who therefore has a chance to apply scrutiny to the reliability and authenticity of the provider.

In general, our goal is that the user of ADVICENET should face no more exposure to security risks than a user normally suffers through routine Internet activity. We believe that in many ways ADVICENET creates fewer risks than existing widely used technology such as Web Browsers, in the forms in which such technology is often used (e.g. with "Cookie Dropping" enabled).

5 Privacy Issues in AdviceNet

Modern PC's contain an enormous amount of information about their users, some of this highly sensitive. It is very important for the acceptance and usefulness of ADVICENET that we aim at the goal

Information on the machine stays on the machine.

In accessing advice and subsequent evaluation of relevance clauses we aim at two key principles

1. *Subscriber Anonymity.* No one should have a systematic way to know who is subscribing to a given advice site; in the standard ADVICENET system, it is not consider acceptable behavior to require the names and addresses of subscribers.

2. *No Feedback.* No one should have a systematic way to know which advice is relevant on which machines. There is nothing in the evaluation mechanism which can provide information about the user's machine to any other party.

Point 2 is particularly worth examining. The lack of feedback remains true *even if a malicious party has published libraries which communicate using the Internet.* Because the RELEVANCE language has a strict object oriented definition, and all object references lead to function calls with literal constants, it is not possible for a relevance clause to evaluate a function with an argument that is the result of an expression. Therefore even security holes cannot lead to publication of private information.

We aim for no more exposure to privacy risk than a user normally suffers through routine Internet activity.

6 More On Security and Privacy

The main ingredient of good security is the user's common sense: he/she should deal only with trusted, responsible advice providers. As described in the document *The AdviceNet System*, there are various ingredients in the system which, supported by trust mechanisms, deliver reasonable expectations of security.

An important ingredient is the division of labor between RELEVANCE language and the rest of the system. The RELEVANCE language does not offer the ability to fix problems, only to identify them. Fixing problems can only be done upon user approval. This gives a layer of insulation and a chance for reflection. The user is free to not approve that certain tasks be done; the user is free to study the situation before heeding an advisory, and in particular to inquire whether the advice itself is still being offered by the trusted site that first provided it, and whether the advice really was provided by that site. By these mechanisms, it is possible to protect trusted sources of advice, avoiding contamination by outside advice, pretending to be from a trusted source.

A further very important component is the recursive nature of the advice system itself. It is possible for an author to publish advice disowning earlier advice from the same author, recommending that it be removed from one's system; it is also possible to publish advice recommending not to heed advice from conflicting or dubious sources.

In short, the ADVICENET system contains a variety of tools addressing security issues; it is not the responsibility of the RELEVANCE language alone.

It must also be admitted that there *are* security and privacy risks in ADVICENET . However, these essentially do not involve the RELEVANCE language.

The greatest *security* risks involve what happens when a user decides to follow through on a piece of advice. At that point, in principle, anything is possible; when the user says "OK" he/she is effectively opening the machine to the advice provider who is then in a position to perform surgery of whatever magnitude.

It is likewise true that user approval poses the greatest *privacy* risks. The application that runs in response to user approval has, in principle, the ability to communicate to the Advice Provider, and there are no real restrictions on what can be communicated. An Advice Provider *can* know if a certain action (update or modification) something was approved by a certain user at a certain time. This can reveal useful information to the provider, since the provider is learning that a complicated condition on the system state is true. Depending on the complexity of the condition, the information revealed may be substantial.

Although the security and privacy risks that would be posed by improper design of the RELEVANCE language are not serious compared to the risks posed by indiscriminate user approval of untrusted follow-through, it is important to pay careful attention to these risks. It can be anticipated that ADVICENET will parse and evaluate many tens of thousands of relevance clauses for every clause that is found to be relevant. Hence many orders of magnitude more work will be done without user intervention than with user intervention. Risks posed by the evaluation process, even if suffered only rarely, might potentially have a numerically far more important role than other risks posed by the system.

It is also important to point out that in special circumstances, one might *not* want the security and privacy features to be available. One can anticipate that in a secure intranet setting, barred from outside access by a firewall, it would be useful for intranet managers to be able to know which machines on a corporate network respond 'Relevant' to certain queries. In that event, it would be useful to disable some of the restrictions we have placed on the language in the design discussed here. Our design goal emphasizes security and privacy, but in certain versions of the final product other emphases will be possible.

One can also imagine that certain advice providers will want to know who is subscribing to advice because they are selling advice and need to make it available only to paying customers. This arrangement violates the stated privacy goals underlying this document. However, while we emphasize privacy goals, we will also enable applications in which such goals are secondary.

7 Properties of the RELEVANCE Language

We now describe important details about the language structure and function. Given the earlier discussion, certain of the information presented here will appear repetitive; it seems best to be absolutely explicit about many details at this stage, and this seems to require repetition.

7.1 Descriptive Language

The RELEVANCE language is used for querying the state of a computer, a highly complex arrangement of software, hardware, and data. It is a description language which describes state rather than a procedural language which describes actions. As the language is not used for traditional procedural tasks (e.g. sorting data, moving data) it is *profoundly* different from a traditional procedural language. There are

- no named variables
- no assignment statements
- no function calls, or at least no explicit function calls with variable arguments
- no loops or conditional execution

Nevertheless, the language is designed to be powerful, in that it is intended to be *highly expressive*; a few words in this language provide access to answers about the system state which would be impossible to obtain in traditional programming languages short of writing hundreds of lines of code and invoking many specialized functions in system libraries.

Certain tasks can be done either in a descriptive language or in a procedural language; the code in a RELEVANCE language will typically look very different than the code for the same task in a procedural language, and the way of productively thinking about how to accomplish the same task will be very different between the two languages. It will be important for some programmers to remind themselves from time to time the main reasons why a relevance clause *should* be very different from a program in a procedural language. Three good reasons are

- Because procedural langauges pose security & privacy concerns;
- Because descriptive languages can be more economical (in terms of code length) for the purposes of writing relevance clauses;
- To make the distinction in purpose starkly obvious to programmers.

7.2 Layered Language Definition

It must be understood that the RELEVANCE language is very open-ended, and that it is built up in layer upon layer of extensions. In this document we only give detailed information about the base 'built-in' layer, with a few hints about other layers.

It must also be understood that the layers interface via object-oriented programming techniques. Hence the capabilities of each additional layer are delivered in packages "plugging-in" new object-oriented deliverables. In the RELEVANCE language these are data types, data properties, data elements, data casts, and operators.

Hence, to understand a completely installed system is to understand the layers which have been installed, and to understand the object-oriented services that each layer provides.

We envision that in a typical installation, these layers will go as follows:

- *Base Layer.* Contains the basic mechanics of clause evaluation: the Lexer, and the Parser for the language syntax, along with a number of Basic Built-in types, properties, elements, casts, and operators. It is expected that the base layer will be the same on every platform carrying ADVICENET .
- *System-Specific Layer.* This consists of Plug-ins which give basic system information about a certain family of computers. For example, one might be able to get the system date and time, the size of various files, the contents of the PRAM, or the names of attached peripheral devices. It is expected that these will all be in common

for a given computer OS family – Win '95, MacOS – independent of manufacturer. It is expected that these will be similar from one OS to another – the object oriented structures and queries being basically the same – but perhaps with naming difference (e.g. "Folder" on one OS might be "Directory" on another).

- *Vendor-Specific Layers.* This collection of potentially a large number of "extensions Layers" would typically be produced by third parties giving special access to the internals of their hardware and software products: one can think of potential authors ranging a span of products from hardware producers (of, say, Cable Modems) to software producers (of say Photoshop & Plug-Ins); to service providers (e.g. America On-Line).

Define
Primitives
Contents

It is expected that 80% or more of the power of the RELEVANCE language will be provided through services offered outside the Base layer.

We do not at the present time have any thoroughly thought-through examples of Class Libraries outside the Base layer. Obviously this is an important next step, which we expect to describe in two documents:

- THE ADVICENET MACOS INSPECTOR LIBRARY. Describes a layer of Macintosh-specific Types, Properties, Elements, and Casts which supplements the base RELEVANCE language with a comprehensive range of services for inquiring about the state of a MacOS computer.
- THE ADVICENET WIN95 INSPECTOR LIBRARY. Describes a layer of Windows '95-specific Types, Properties, Elements, and Casts which supplements the base RELEVANCE language with a comprehensive range of services for inquiring about the state of a Wintel '95 computer.

7.3 Expected Conditions of Use

We anticipate that many advice providers will use the RELEVANCE language under the following conditions.

- A programmer will author advisories containing relevance clauses.
- Relevance clauses will typically use types, operators, elements, properties from existing inspector libraries to obtain their expressive power.
- Very occasionally, the programmer will discover that a key property (say of a hardware product) is not made available through existing libraries, and will author such a property plug-in.
- The programmer's advice will depend on the existence of various plug-ins for its well functioning – both widely available pre-existing class libraries and his own newly-developed plugins. He will have the responsibility to verify that libraries are correctly installed. It is necessary to write meta-advice to verify this.

Thus the programmer has these responsibilities:

- To know the base language and system-specific inspector library;
- To write the relevance clauses for his advice using those inspectors;

- (Occasionally) to extend the collection of existing types and properties;
- To write meta-advice verifying that the language is properly configured to correctly evaluate his advice.

8 The Object Model

While a computer language involves both syntax and a collection of semantics for manipulating data structures, we have chosen in this document to begin with the role of data structures and semantics, and later to discuss syntax.

As discussed earlier, the RELEVANCE language is object-oriented, which means in our case that

- The language is *strongly-typed*: every object has a specific type, and operators and properties may only be applied to objects of appropriate types. (In some non-object-oriented languages, it is possible to misuse functions, applying them to data for which they cannot work properly. This is not possible in the RELEVANCE language).
- The language is *polymorphic*: the meaning of an operator or property reference depends essetially on the type of the direct object. This is also called *overloading* of operators & properties: the interpretation of an operator or property depends heavily on the data to which it will be applied.

The language differs from some other OOP languages in that the concept of encapsulation is de-emphasized. There is no distinction between private, public, and protected data. Or, more precisely, all data are public.

In the current revision of this document, Version α .10 , the important object-oriented notion of *inheritance* will not be discussed; although it is expected to ultimately be included in the system. See Section 8.4 below.

8.1 Objects and Methods

An *object* in the RELEVANCE language may be thought of as a reference to a physical object, such as a *file* or *SCSI device*. It is implicitly a data structure, to which can be applied access methods made available in the RELEVANCE language.

Every object has a *type*, and each type has associated with it several *accessor methods*. The language has four base types, which are essential to the workings of the language interpreter, and many extension types which are essential to the performance of useful work.

The base types are

- *Integer*. A signed 32-bit integer.
- *String*. A variable length character string.
- *Boolean*. A single bit.
- *World*. An abstract type, representing the computer of which the language is running.

The RELEVANCE language accessor methods can be thought of as giving answers to structured queries about objects. They are strongly-typed: an accessor may only be applied to a type for which a definition of that accessor for that type has previously been registered.

Accessors themselves have various classifications:

- *Properties*: methods which inspect an object and return a value.
- *Elements*: methods which inspect an object and under the control of a single parameter, return a value. Often these are components of the object's structure and hence are called elements. *Element-by-iIndex* takes an integer; *Element-by-Name* takes a string.
- *Operators*: methods which inspect an object (unary operators) or a pair of objects (binary operators) and return a value. These are invoked syntactically by the usual arithmetic and relational operators as commonly encountered in high-level languages; but there is no requirement that actual semantics be familiar. The interpretation of `a + b` need not resemble the usual interpretation – if a and b are not of integer type.
- *Casts*: methods which inspect an object and return a new object with a different type or, in some cases, the same type but a different format.

The purpose of the type-method system is to make available powerful query and manipulation tools but only under strong-typing restrictions that inhibit misuse.

8.2 Under The Table

The RELEVANCE language object system is intrinsically connected with the object system provided by C++.

Each Type, Operator, Property, Element, and Cast defined in the language corresponds to a class, an operator, or a public member function in C++

In fact, extensions to the base set of types are provided by writing appropriate C++ code; see the document THE ADVICENET INSPECTOR API for an explanation how this is done.

We can view the library of extension types as providing a rich library of C++ routines for querying the system state and for manipulating the answers that get returned.

We can therefore view the RELEVANCE Language as a tool for providing access to such a library while imposing strong-typing restrictions which inhibit abuses.

8.3 The Object Universe

The bulk of the functionality of the RELEVANCE language is provided by the extension types which have already been mentioned. These can be conveniently organized as a mathematical graph structure, in which *nodes* represent *types* and *arrows* represent *accessor methods* which return properties, elements, or casts of those types.

The structure is rooted at `World`, which has many properties. On the Macintosh, a few of these are:

Properties of World

- System Folder. Returns an object of type *Folder* associated with the system folder.
- Boot Drive. Returns an object of type *Volume* associated with the boot volume.
- Processor. Returns an object of type *CPU* associated with the computer.
- Printer. Returns an object of type *Printer* associated with the currently chosen printer.

- Monitor. Returns an object of type *Monitor.*
- IPAddress. Returns an object of type *IPAddress.*

We can list a few elements as follows:
Elements of World

- SCSI Bus *Integer.* Returns an object of type *SCSI Device* associated with the specified SCSI Bus slot.
- Serial Port *Integer.* Returns an object of type *Serial device* associated with the specified SCSI slot.

It should be clear that, in general, there is a close connection between methods and the types that they return. Often these even have the same name. Remember, though, that the method that has name 'Processor' obtains a property of *World*, not of *Processor.*

The methods applicable to *World* generate a series of types with their own methods; those methods in turn generate other types. For example, an object of type *Volume* has properties *Driver* and *File-Allocation-Blocksize* and *Capacity* and *FreeSpace*, while an object of type *CPU* has properties *Name*, *Speed*, *FPU*, *MMX*, and so forth.

Some of these objects are of atomic type: *Speed* when applied to an object of type *CPU* yields an object of integer type; while *Volume Name*, when applied to an object of type *Volume* yields an object of type string. However, some of these objects are of new type: witness *Driver*, which is an object with properties such as *Vendor* and *Version.*

In a future version of this document we expect to work out in detail all the types and methods associated with two areas of special interest:

1. Internet Settings example
2. Volume examples

8.4 The Evolving Universe

The object model we are describing here is still under development. Two major enhancements to the system described here are currently under development.

- *Inheritance.* A basic platform-independent type (but with platform-dependent implementation) is *File.* This is an object with the following properties on MacOS:
  - Name. Object of type *Filename.*
  - Creation Time. Object of type *Time*
  - Modification Time. Object of type *Time*
  - Length. Object of type *Integer.*
  - Checksum. Object of type *Integer.*
  - Parent Folder. Object of type *Folder.*
  - Version. Object of type *Version.*
  - Type. Object of type *MacFileType.*
  - Creator. Object of type *MacFileCreator.*

Under MacOS, there are also files with special properties, and hence having their own specialized types: *Application*, *Control Panel*, *Extension*. For example, an application has all the properties of a file as well as a *Partition Size*, *Recommended Partition Size*, *Minimum Partition Size*, and *Bundle Bits*. A control panel has all the properties of file, as well as the property *Enabled* (a Boolean).

In traditional OOP systems, one would handle such issues by creating a base type *File* and derived types *Application*, *Control Panel*, etc. The derived types accept all the same accessor methods as the base type *File*, and objects of type *Application*, say, accept also the accessor methods for that type, such as *Partition Size*. We expect that the RELEVANCE language will ultimately do this, though the details remain to be worked out.

- *Plural Properties*. It is important for certain purposes to know that there only be one of a certain object in existence, while for other purposes, it does not matter if there be one or more than one instance matching a reference.

  As a convenient way to make it possible to verify either circumstance, the RELEVANCE language will offer plural properties. For example, the property *application* is singular and the property *applications* is plural. The distinction is this

  - `application "Netscape"` asserts the existence of *exactly one* application named Netscape in the scope. Clause evalaution will generally fail if there is no such application or if there are more than such application (see Section 9.2) for information about the mechanism behind this.
  - `applications "Netscape"` asserts the existence of *one or more than one* application named Netscape in the scope. Clause evaluation will generally fail only if there is no such application (see Section 9.2).

  The existence of singular and plural properties means that phrases have (by inference) singular or plural characteristics. Accomodating this will require certain modifications to the grammar and semantics of the language, which remain to be worked out.

8.5 Describing the Universe

In order to keep track of all the types and methods available in ADVICENET , we will make available a *Type Inspector*, a specialized dictionary indicating all the available types and their associated methods and the results of thse methods.

This dictionary will be rendered in two different ways. In an on-line version, it will offer a hypertext-format dictionary describing each relevant term and the associated definition and relationships. This will be reminiscent, for some readers, of the Class Dictionary in AppleScript [1].

In a printed version, it will take a standard format reminiscent from typical system manuals for UNIX and related operating systems. Roughly speaking, we expect a single-page description of each type, taking the form

Typename – *1-line synopsis*

Description. - 1 paragraph of text describing the general role of this type, and important considerations for those making use of it.

Contained-in. – Name of library that installs it. URL of library source. Current version of that library.

Properties.

Prop1 Value-Type. Description of what the value is supposed to be, and the type that it takes.

...

Elements.

keyword1 name Value-Type. Description.

keyword2 integer Value-Type. Description.

...

Unary Operators.

unop1 Value-Type. description.

...

Casts.

as **Typename1** description

...

How-reached. *Description of Types from which this type is reachable as the value of a property or element.*

property1 of Type1

...

Related Types. *Description of Types which have intimate connections with the current one, either*

Type1 Reason.

...

Subtleties. *Hairy points to keep in mind.*

1. Note1.

...

9 The Evaluation Model

We now turn to rules for successful interpretation of clauses in the RELEVANCE language.

9.1 Abort, Retry, Fail

The basic purpose of the RELEVANCE language is to successfully lex, parse, and evaluate a single expression, resulting in a value of type *Boolean*. The resulting value is passed to the ADVICENET advice reader for further processing.

Of course, there will always be cases where a RELEVANCE clause is malformed or otherwise unacceptable. In order to deal with that situation in an organized fashion, the lexer/parser/interpreter follows this rule.

> A RELEVANCE Clause will lead to an advisory being declared relevant *only* if, the clause is *intelligible*, *successfully evaluates* to a result of type *boolean*, and is *true*.

Let's explain this rule in more detail:

- Expressions can be *Unintelligible* or *Intelligible*. *Unintelligible* expressions occur for one of three reasons.
  - *Lexical unintelligibility*. For example, the expression contains bizarre unquoted characters, such as $.
  - *Syntactic unintelligibility*. For example, the expression contains contains disallowed usage, such `++name`.
  - *Run-time unintelligibility*. For example, the expression refers to an unknown property (e.g. `exists solarplexus of file`).
- Intelligible expressions can *Succeed* or *Fail* to evaluate.
  - *Arithmetic Failure*. 65536*65536 overflows type *integer*. $\rightarrow$ NSO
  - *Reference Failure*. `picture resource 101` doesn't exist.
  - *System Failure*. There is an error reading the hard drive.
- Successful evaluations can be *Boolean* or not. Examples of types which are not Boolean: *integer*, *IPAddress*, etc.

9.2 Exception Handling

The RELEVANCE language interpreter creates two special objects to indicate evaluation failures.

- NSO, which stands for 'No Such Object'.
- TMO, which stands for 'Too Many Objects'.

For example the expression `length of file "foo"` evaluates to NSO if file `"foo"` doesn't exist.

NSO *generally* but not always causes expressions to FAIL to evaluate. There are at the moment two exceptions:

1. `exists(NSO)` evaluates successfully.
2. `exists folder whose ( length of file "foo" of it is 8 )` evaluates successfully if SOME folder contains a file `"foo"`, even though many folders do not.

The expression `application "Netscape"` evaluates to TMO if there are *two or more* applications which are instances of Netscape.

TMO *generally* but not always causes expressions to FAIL to evaluate. There are at the moment two exceptions, analogous to the exceptions for NSO discussed above.

10 RELEVANCE Language Syntax

We finally attempt a description of the syntax of the RELEVANCE language!

10.1 Simple Examples

1. Existence of a certain application.

```
X-Relevant-When: exists application "Photoshop"
```

2. Comparison of version numbers.

```
X-Relevant-When: exists Control Panel "MacTCP" and
                 version of Control Panel "MacTCP" is version "2.02"
```

3. Compare modification dates.

```
X-Relevant-when: exists Photoshop PlugIn "Picture Enhancer" and
                 modification time of Photoshop PlugIn "Picture Enhancer" is
                 greater than time "10 January 1997"
```

4. Examine Control Panel Settings.

```
X-Relevant-When: exists Control Panel "ConfigPPP" and
                 Transport Mechanism of Control Panel "MacTCP" is equal to "SLIP"
```

We now turn to a more systematic description of the language; we consider in turn: Lexical analysis, Syntax, and Semantics.

10.2 Lexemes

The lexical analysis of the string breaks up the input string into tokens consisting of basic elementary inputs.

In the following, we will denote literal keywords by enclosing them in curly braces {`like this`}, and we will denote general Lexeme Classes by [class name].

Lexical tokens come in one of the following basic categories.

[String] A string of printable ascii characters enclosed in quotation marks (").

[Integer] A potentially signed string of decimal digits.

[Minus] The character -.

[SumOp] The characters +-.

[PrdOp] The characters */.

[RelOp] The character sequences = > >= <= !=.

[Phrase] A sequence of one or more unquoted words.

These categories are mostly unexceptional, and will be familiar to programmers from work with other computer languages.

10.2.1 Phrases

The most non-standard aspect of the language is in the definition of token type [Phrase]. Formally, a Phrase is a string of words; a word is a string of characters beginning with a letter. Any Phrase can be further decomposed into several occurrences of [Reserved Phrase], with, intervening word sequences (by definition) called [Ordinary Phrase].

Here is a partial list of
Reserved Phrases.

- as
- and
- containing
- ends with
- exists
- is
- it
- number of
- or
- remainder
- starts with
- whose

These phrases are all built into the base layer of the language, and have purposes associated with the base functions of the language; hence they are reserved from other use.

Here is a partial list of
Ordinary Phrases.

- Control Panel
- Photoshop PlugIn
- Modification Time
- MacFileType
- MacFileCreator
- SCSI Bus
- Shared Disk
- CD ROM

These are all phrases that describe system-dependent concepts, and so they are associated with the System-specific or Vendor-specific layers of the language. The collection of ordinary phrases is not fixed in advance, and can be exepected to grow with time as more inspectors are added to the language.

We note that one has *Run-Time Unintelligibility* when the lexer extracts an ordinary phrase from a string of words and cannot find that phrase in the current collection of installed libraries.

10.2.2 Special Notes

1. Strings are simply quoted sequences of ASCII characters.
   - There is no a-priori length limit on strings. It is a *System Evaluation Failure* if the lexer encounters strings so long that they cannot be stored in available RAM.
   - RFC 822 imposes length restrictions on strings.
   - There will be a mechanism for continuing strigs across lines.
2. Escape sequences will be specified, but have not yet been defined. Most likely usage is to specific ASCII codes explicitly by $\backslash D_1D_2D_3$ where the D_i are decimal digits, although hexadecimal and other coding schemes might be offered. However, issues of accents and other items that appear in connection with foreign character sets will have to be carefully studied.

10.3 Formal Grammar

In the table below, we denote concepts defined in the left-hand-side of Grammar Productions by ⟨name⟩

```
<Goal>          :=  <Expr>

  <Expr>          :=  <Expr> {or} <AndClause>   | <AndClause>

  <AndClause>     :=  <AndClause> {and} <Relation> | <Relation>

  <Relation>      :=  <SumClause> [RelOp] <SumClause> | <SumClause>

  <SumClause>     :=  <SumClause> [SumOp] <Product>
                  |   <SumClause> [Minus] <Product>
                  |   <Product>

  <Product>       :=  <Product>   [PrdOp] <Unary>
                  |   <Unary>

  <Unary>         :=  [Minus] <Unary>
                  |   [UnyOp] <Unary>
                  |   <Cast>

  <Cast>          :=  <Cast> {as} [Phrase]
                  |   <Reference>

  <Reference>     :=  [Phrase] {of} <Reference>
                  |   [Phrase] [string]    {of} <Reference>
                  |   [Phrase] [integer]   {of} <Reference>
                  |   [Phrase] <Restrict> {of} <Reference>
                  |   [Phrase] [string]
                  |   [Phrase] [integer]
                  |   [Phrase] <Restrict>
                  |   [Phrase]
                  |   {exists} <Reference>
                  |   {number of}  <Reference>
                  |   [string]
                  |   [integer]
                  |   {it}
                  |   ( <Expr> )

 <Restrict>       :=  {whose} ( <Expr> )
```

10.4 Special Language Constraints

There are two very important special constraints imposed by the syntax of the language which we now discuss. Each one can be described in terms of what the RELEVANCE language does not allow which programmers may be familiar with due to experience with procedural languages.

10.4.1 Lack of variables in function calls

For the lack of a better terminology, we have chosen to use (archaic) procedural language to describe this special feature of the language. The usage doesn't quite fit because the RELEVANCE language really has no function calls.

Still, we may think of "Element-by-Name" as implicitly invoking a certain function with two arguments the first being an object, the second a parameter that selects the precise object. For example:

```
                 Parent of application
file "Read Me" of folder "Photoshop"
```

might be coded in a procedural language as

```
for (i=0; i<NVolumes; i++){
   Volname = Volume[i];
   FolderRef = findfolder(Volname,"Photoshop");
   if(FolderRef != 0){
       FileRef    = getfile(FolderRef,"Read Me");
       break;
   }
}
```

Here `folder "Photoshop"` asks for an element-by-name of the World, which is similar to calling, for each mounted volume, the procedural language function `findfolder(volname,name)` with appropriate parameters to have it search a certain volume for a certain folder.

The key point is that the syntax of the RELEVANCE *language forces you to ask for specific elements-by-name using string* **constants** *for the selectors. It is impossible to perform the analog of calling such a function with a computed name.*

Thus, one can not say

```
file (filename of file 3 of volume "Shared Disk") of
folder "Photoshop" of volume "Private Disk"
```

This is a potentially useful script: one looks on one of volume for a file and looks for a file by the same name on another volume. *It is forbidden syntactically to do such things in the* RELEVANCE *language.*

The rationale for this taboo is the existence of security problems which would inevitably appear if it were tolerated. We expect that for many advice authors there will be a period of psychological adjustment as they adapt to this restriction.

10.4.2 Lack of Nested Loops

Since there are no variables or assignments in the RELEVANCE language, one may wonder at first how one could obtain the equivalent effect of iterating across a list of folders or files to check a condition. In the RELEVANCE language *loops a single level deep* can be handled by the use of the `whose` primitive, and the associated `it` reference.

For example, suppose we wanted to find out if the user's Eudora folder has any mail digest files which were last modified before the ship date of the current release of Eudora.

```
exists file of folder "Eudora"
  whose ( modification time of it is less than time "12 January 1997" )
```

Here the `whose` clause is causing the implicit iteration over all files in the Eudora folder. It is useful to think of `whose` as creating a function with the single formal parameter `it` and then calling the function once for each file in the indicated folder.

On the other hand, suppose we wanted to find out if the user's Eudora folder has any mail digest files which are not accompanied by table of contents files. We might think of writing

```
exists file of folder "Eudora" whose ( not exists (file (prefix of it & ".cnt")))
```

This pseudo-fragment will not work because it specifies the forbidden usage "file (computed-result)" (see previous subsection)

We might also think of writing

```
exists file of folder "Eudora"
     whose ( suffix of it-1 is ".mbx" and
        not exists file of Folder "Eudora"
            whose (prefix of it is (suffix of it-1 suffix of it-2 is ".cnt") )
```

This psuedo-fragment is an attempt to make nested loops; we would like to have `it-1` be the 'local variable' in the 'scope' of the 'outer whose' and `it-2` be the 'local variable' of the 'scope' associated with the 'inner whose' . Note however that these are vain impulses: they are not allowed syntactically.

11 To Probe Further

- Relation to AppleScript.
- Readings on OOP.

12 Appendix 1. Properties of World

13 Appendix 2. The Base Library of Types and Accessors

References

[1] The Tao of Apple Script.

[2] The Tao of Objects.

[3] THE ADVICENET SYSTEM.

[4] THE ADVICENET SITE DEVELOPER'S MANUAL.

System changes since last TIME!

[5] The AdviceNet Inspector API.

[6] The AdviceNet Mac OS Inspector Library.

[7] The AdviceNet Win95 Inspector Library.

LAST

EXHIBIT E

The AdviceNet System for Automating Technical Support

Universe Communications, Inc.
2180 Dwight Way, Suite C
Berkeley CA 94704

12/10/97

Abstract

The ADVICENET system allows technical support organizations, such as software and hardware providers, Internet service providers and corporate IT departments, to support very large communities of client personal computers and their users. It allows such businesses to pro-actively avoid problems on client PC's before they arise and automatically repair problems on client PC's as they arise. It is based on novel use of Internet technologies, and novel tools to describe and query the state of PC's. It fits naturally in the existing work pattern of technical support organizations.

ADVICENET delivers *advice documents* from *advice providers* to *advice subscribers.* Advice documents describe potential problems which users may be experiencing as well as the solutions of those problems. Problems are described in a formal language which is read automatically by copies of the ADVICENET Advice Reader running on each individual computer. Advice documents typically reference an applet solving the given problem on the given machine or describing user actions which will fix the problem. The advice reader can tell if the computer it is running on is in need of any given piece of advice and when it determines there is a need, will offer the user to apply the solution automatically.

ADVICENET is an inexpensive and supremely efficient mechanism for communicating solutions of potential technical problems. An advice provider need only prepare an advice document and place it at an advice site to ensure that the problem description and recommended solution begin immediately and automatically to diffuse over the Internet to thousands or even millions of users. Many advice providers will recognize this to be an overwhelming improvement on existing methods of problem resolution/solution delivery, such as mailings or 1-800 phone support. Many technical support organizations have recently mounted massive efforts to develop knowledge bases of problems and solutions; they will find that ADVICENET allows them to effectively use that knowledge in a pro-active way.

ADVICENET is also attractive to individual users. ADVICENET solves problems on the user's computer, while rigorously avoiding unnecessary intrusions on the user's time and attention. Advice is brought to the user's attention by the ADVICENET system only when the ADVICENET reader recognizes that the user is having the indicated problem. The user need never know about the ADVICENET system until his machine has the symptoms of a recognized problem, and the user need only 'click ok' in order to fix the problem. In addition, a central feature of ADVICENET is the ability to protect the user's security and privacy.

This document describes in a broad overview.the basic components of the ADVICENET system and the services and benefits they provide.

19 PAGES

Contents

1 Introduction

This document describes, in broad overview, the ADVICENET System, and the role it can play in automating technical support services. The introduction describes the extremely complex and costly world of modern PC's. Later sections develop a picture of the ADVICENET system and how it addresses the key challenges of the current situation.

1.1 Making Computers Work

There are now half a billion computers on earth; within a few years, this number will approach one billion. On any given day, tens of millions of computers crash; as a result, millions of people worldwide lose valuable time, data and money. The object we call a 'personal computer' is, despite the friendly name, a highly complex collection of interacting systems, with many software applications, hardware items, and networking protocols installed and able to interact in unforeseen ways. Certain combinations of products are troublesome; certain products work well only when certain system parameters are set appropriately, or when certain files are installed in appropriate locations. Quite innocent actions by users – moving files, changing settings, closing applications – can literally render a personal computer unusable.

The complexity of this environment is continually increasing, and this gives ever more opportunities for trouble to arise. There are now tens of thousands of hardware and software products available for use with the computer, and more are being introduced every day. As a result, it is impossible for hardware and software producers to thoroughly test all combinations of installed products, or all variations on computer parameter settings. Moreover, the problem is not necessarily 'bad programming'; the environment is so complex, that no matter how 'good' the programming is, problematic interactions will be discovered after a product is released to the public. One can expect that there will always be an unavoidable component of 'computer problems' caused by the complexity of the environment and that in sheer numbers of people affected, this number will grow ever larger as computers become more integrated into ever wider ranges of human activity.

In a strong sense, computers are more trouble-prone than other useful items in our everyday lives: by any standard, cars, photocopy machines, and other similar objects are far more reliable than computers. Yet by and large, computer support is handled in the same fashion as far simpler technologies – through 1-800 phone calls to hardware and software manufacturers, through a network of sales representatives and product dealers, and through a network of systems integration consultants.

The problem is too large and too difficult to be handled in this fashion. The usual approaches for making computers work well have not been designed to cope with the explosively growing size of the number of computers and users, or with the increasing number of potentially problematic interactions. In fact, continuing current trends, the day is coming when there will not be enough 1-800 telephone staff, nor enough dealers and systems consultants, to deal even minimally with all the support problems that will arise.

1.2 The Financial Burden

Keeping computers running is a costly business for everyone concerned. While there is no single figure that captures the full dimensions of the situation, the picture that emerges by combining many different sources is one of tremendous expense in response to the tremendous intrinsic complexity.

- *Burden on Corporate Budgets.* The Gartner Group has championed the concept of 'Total Cost of Ownership' as an important figure for management to track and respond to. It estimates that the three-year cost of owning a single personal computer in a corporate setting exceeds $44,000 [4]. The initial investment in the hardware and software used on the machine is dwarfed by other costs of ownership. Gartner estimates that 27% of the Total Cost of Ownership is due to Technical Support, which all by itself exceeds the capital cost of initially acquiring the equipment and software (only 21% of Total Cost of Ownership).

- *Burden on Moderate-Sized Businesses.* The idea of *not maintaining* IT departments and Help Desks is not really an option, even for moderate-sized businesses. According to a report by Nolan Norton and Co., businesses which operate PC's without a help desk support structure suffer costs of between $6,000 and $15,000 per year – the cost of lost productivity for employees who can't get their work done or must interrupt other workers. per PC

- *Burden on Small Businesses.* Small businesses are simply too small to have IT departments and Help Desks. Instead, support for small office computing is conducted by outside contractors, systems consultants who often spend time solving rather minor issues for customers. There is reportedly a dramatic shortage in major metropolitan areas of systems consultants who can help maintain small office computing systems, with available consultants charging fees running at $125.00/hour even for basic tasks. This cost seems particularly high when the maintenance tasks being attempted are often simple technical support issues.

- *Burden on Software Producers.* According to the Association of Support Professionals [5] the 1997 median cost of support for a medium to large software developer is 8% of revenues. This varies by type of product, with developers of high-end products spending as much as 20% of sales on support.

- *Burden on Large Software Producers.* Microsoft answers 22 million technical support calls per year. In order to keep up with the press of user queries, it has invested $500 million over the last three years to develop new Technical Support infrastructure [7].

These are just a few examples of cost reports which are publicly available. They all point to the great cost and complexity of existing tech support arrangements. This would be bad enough in a slowly changing world. But unfortunately, the world is rapidly changing, and the computational universe is rapidly expanding. The financial burden of technical suppport issues will only become worse as the number of computers continues to expand.

2 There Must be a Better Way

Much of the complexity and frustration currently experienced by millions of computer users *could be avoided*. We maintain that the frustration and complexity are caused by a *communications problem* which technology can solve.

2.1 Support is a Communications Problem

We can identify two classes of actors in the process of computer maintenenance and support:

- *Experts* - who know about many potential problems and could help users avoid or solve those problems; and
- *Users* - who need to get solutions but can't hope to easily find just the solution they need for the problem they have.

Experts, in this viewpoint, are individuals with special knowledge about specific situations and methods to fix them. They could be working for any of a large variety of organizations. Typical examples include

- *Software Providers.* Individuals in software development and in technical support who know of problems that users of their products will experience under certain conditions.
- *Hardware Providers.* Individuals in harware development and in technical support who know of problems that users of their products will experience under certain conditions.
- *Corporate IT managers.* Individuals in large organizations who know about situations special to their own corporate mission and their own mix of applications and hardware.
- *Internet Service Providers.* Internet Service Providers who need to insure that their users connect smoothly to the network and obtain access to special services they are entiteld to.
- *User Groups.* Individuals managing the sharing of large bodies of advice about the care and maintenance of specific machines/ software products/ hardware products.

In each case, the organization will know a great deal of information about problems likely to be experienced by the user population, and a great deal about how to avoid and to fix such problems. In many cases, this information is informally recorded – residing simply 'between the ears' of the experts; in other cases, it is recorded in manuals and technical reports; in still other cases it is formally documented in technical knowledge bases which the organizations have compiled patiently over years of effort.

A given user might potentially be interested in solutions provided by any or all of these types of experts, meaning that dozens or even potentially hundreds of experts in the world at large might know of important situations and corrective remedies on that given user's computer. For the user to benefit from the existing expertise, the user must communicate with any or all of these experts, share with them a description of his concerns/problems, and obtain from them solutions and partial solutions. So the problem of computer maintenance and reliability can be viewed as a communications problem.

2.2 Communications Methods Used Today

Unfortunately, existing approaches for joining the key actors together are clumsy and inefficient. They require great expenditures of time and money, often for the purposes of communicating trivial amounts of useful information.

2.2.1 1-800 Technical Support.

For problems which are so bad that the user perceives that they require immediate attention, the standard solution is for the user to call the technical support organization associated with whatever hardware or software vendor the user suspects may know the solution.

From the organization's point of view, the process has severe drawbacks

- *Costly Misdirection.* Frequently user calls are misdirected. For example, the appropriate organization for the user to contact is different than the organization he has chosen to contact. Or, the problem identified by the user may not be a problem at all, but simply a quirk or misunderstanding. Nevertheless, it can take a great deal of time for the support technician to see that a different organization should be handling the problem.

- *Costly Diagnosis.* It is frequently difficult for the technical support professional to diagnose the possible problem. He does not have direct access to information about the configuration of the machine, and often must spend a great deal of time asking elementary questions of the user in order to understand the specifics of the problem and of the PC configuration. This is particularly so when one considers that often the problem being identified is relatively straightforward ('you need a new version of the program', 'you need to correctly set your TCP/IP information'). The resulting expense is out of all proportion to the underlying amount of information being obtained.

- *Costly Delivery.* Even when the process produces a clear diagnosis of the problem, and so has clear benefits, the costs incurred – telephone and salary - may be unreasonably large. This is especially evident when one considers that often the solution to a problem is a relatively trivial one ('move such and such a file'; 'change such and such a control panel setting'; 'get this update'). The expense of solving such problems is out of all proportion to the underlying amount of intrinsic work required to solve it.

- *Costly Repetition.* As a general rule, few problems that arise are unique. Most problems which generate tech support demands are experienced by many users. *Many tech support calls are destined to be repeated a thousand times over,* as user after user experiences the same problem again and again.

The process is also inefficient for users:

- *Frustrating Waits.* There is often a very long wait on the phone line before reaching a technical support professional. At major hardware and software manufacturers today, waits of 15 minutes to half an hour are common.

- *Frustrating Searches.* A common outcome of a technical support call is the technician's instruction that the use's problem really should be handled by a different organization. This means that the user bounces around, from organization to organization, searching for a friendly ear.

- *Failure to Solve.* Often the technical support professional is not able to reach a clear identification of the problem. In other cases, the user is not able to understand or utilize the information provided by the technical support staff.

2.2.2 Printed Matter

Many problems experienced by users could be avoided if they had a complete knowledge of information already provided in various manuals and README files. However, by and large, computer manuals do not get read, and neither do README files.

Even for computer-literate professionals, it is often difficult to search through the information contained in such resources and find out what, if any, of the many messages there might be applicable either to the user's machine, or to the user's current problem. The complexity of the problem really outstrips the linear nature of printed information.

2.2.3 Web Sites.

The most dramatic development in technical support capabilities over the last decade is the use of web sites to facilitate technical support. Web sites are already widely used to distribute software updates and, in a fraction of large organizations, they are used to make available client-searchable technical knowledge bases.

For sophisticated users particularly, this can be a very useful delivery mechanism. For less sophisticated users, the mechanism is less effective, because the diagnosis of a problem often requires a great deal of context which the user is unable to provide.

The web site approach to problem resolution has not proved itself financially attractive in many cases; the Association for Support Professionals has quoted a cost per web site support transaction at $28.12, which is not really cheaper than a telephone support transaction. Obviously, the point is that relatively few people are using such knowledge bases; they are unable to find what they need, or they simply prefer the telephone transaction; hence the costs of developing a site must be amortized over relatively few transactions.

Even for sophisticated users however, the fact that there are so many different 'experts' who could be providing information means that in general there are many web sites that might potentially need to be consulted in order to solve a problem. The result is an unreasonable search and communications burden.

Finally, even for distribution of updates, one can question whether web sites are really effective. By and large users do not know when an update is available and do not install it until told to do so by the tech support organization. If the user updates only because a phone call to technical support, the main cost of tech support – the phone call – has not been avoided or ameliorated.

2.3 Summary of Communications Obstacles

We could of course say more! In general, existing mechanisms of problem resolution suffer from the following shortcomings:

- *Excessive investment of time by Experts.* Experts with the knowledge to solve problems spend little of their time using their knowledge and most of their time attempting to communicate – i.e. to get from the user the information needed to diagnose and prescribe.

- *Excessive investment of time by Users.* Users trying to access expertise spend little of their time actually receiving expertise and most of their time attempting to communicate – i.e. to figure out the right support provider to help with a certain problem, to understand questions posed by technicians and get the answers to those questions, to relay to the technician the answers about their machine or (in existing automated approaches) to form search queries and sort through voluminous query responsae.

- *Excessive reliance on sophistication of users.* Existing approaches rely overwhelmingly on the ability of users to identify problems, to understand, to analyze, and to apply expertise.

At a higher level, we can criticize existing arrangements in three ways.

- *Passivity.* The existing approach to technical support is not pro-active: it waits for the user to have a problem and then to call the expert to obtain the fix. When there are known problems, affecting even a small fraction of a user base, we have a sadly predictable result: the problem is addressed only by waiting for literally thousands of phone calls from users and spending thousands of man-hours to connect experts with the users to fix the problem.
- *Ineffectiveness.* The existing approach is often ineffective, because it can't often obtain from typical users the information which would be necessary to diagnose the problem and obtain a solution.
- *Wastefulness.* After a potential problem and solution are identified, the problem continues to generate tech support demands, over and over, as new users experience the problem.

3 There is a Better Way

Univers~~al~~e Communications, Inc. has developed the ADVICENET system to facilitate the avoidance of, and speed the resolution of, a large fraction of typical computer support problems. Our customers, typically large organizations with large numbers of computers and users to support, will find our approach a very cost-effective way to increase the reliability and functionality of large numbers of computers, and to reduce the overall rate of escalation of support costs.

3.1 Technical Support in a New Key

Using our new approach, tech support will begin to exhibit these characteristics:

- *Problems are addressed pro-actively.* Experts who are aware of a potential situation and its solution can publish the information and have it widely available for use as needed.
- *Information is distributed automatically* via Internet, avoiding the cost and delay of physical distribution.
- *Automatic Diagnosis.* Wherever possible, the presence of problematic conditions will be ~~made~~ automatically detected, without intervention of either Expert or User.
- *Automatic Solutions.* Wherever possible, solutions are delivered automatically without the need for user intervention.

The whole process is much more effective for the support organization. Many calls to tech support are avoided altogether, and others are made drastically shorter. Expert involvement to communicate with the user simply to identify the user's situation prior to diagnosis of the problem is avoided. Expert intervention to make the diagnosis is minimized. Many 'small' problems affecting only a small fraction of a large organization's user base, but affecting collectively many tens of thousands of people, that previously consumed significant tech support efforts, can now be solved with very little effort on the part of tech support personnel.

ADVICENET allows an organization to adhere to the principle

Problems Get Addressed Just Once.

Once a problem has been identified and its solution is understood, the tech support provider can immediately publish the problem description and solution in a way which lead to the avoidance of further calls of a similar nature. There will no longer be a need to passively wait for potentially affected users to experience a problem and call tech support in order to get a solution.

The process will also work better for users:

- *Automatic Matching of Providers and Users.* The provider who knows how to solve the given situation is connected precisely with the user who is in that situation and needs the provider's help.
- *Automatic Diagnosis.* Wherever possible, the presence of problematic conditions is determined automatically, without intervention of the User.
- *Automatic Solution.* Wherever possible, the solution of problematic conditions is provided automatically upon User approval.

The new approach will avoid the need for sophistication on the user's part, and will also avoid the need for time-consuming involvement by the user, as well as the need for disciplined, routine efforts on the user's part. And it will work better than what he can get today – the user will routinely be receiving the *best* diagnosis that an expert organization has to offer, and the user will routinely be receiving a true automatic solution rather than a lengthy prescription for work by the user.

3.2 The AdviceNet System

Universe Communications, Inc. # offers a package of tools and services which allow the Technical Support Organization (TSO) to provide pro-active, automated support. In broad over-view, there are the following components:

- *Advice Site.* The TSO offers an advice site, essentially a standard web-server connected to the Internet, at which ~~various~~ various documents are published.
- *Advice Documents.* These documents are descriptions of potential problems on PC's in a formal language – the AdviceNet Relevance language – and descriptions of working solutions. They are developed and published by the TSO.
- *Advice Reader.* Members of the TSO user community, distributed across the internet, install the AdviceNet Advice Reader, and direct the reader to subscribe to the TSO's Advice site.

The AdviceNet system sets up a distribution channel whereby users automatically obtain information about potential problems they may be having. The channel makes the communication of potential solutions painless and efficient.

In broad terms the system works as follows.

- On a regular basis, perhaps daily, the Advice Reader automatically checks the TSO's Advice Site to see if any new advisories have appeared. If so, the Advice Reader downloads the new advisories, and installs them into the advice database.

- On a regular basis, perhaps hourly, the Advice reader literally reads the advisories in the advice database and compares them with the current state of the PC. If any of the advisories mention a condition which applies to the user at that moment, they are marked *Relevant.* This means they are applicable to the computer in question, and also of active and not just potential interest. The user is in a condition that the advice provider has determined ~~is problematic.~~ needs attention
- If there are any relevant advisories, the Advice Reader attempts to get the user's attention. The user, on opening the advice browser, will be presented with a list of relevant advisories and may inspect them one by one. This is analogous to the process of reading e-mail.
- An individual advisory will explain to the user the situation which has been identified, and will recommend an action to correct it. In most cases, there will be an 'OK button' which the user can push in order to initiate an automatic repair.

The system is easy to use – for all involved:

- The Advice Provider (TSO) need only author the Advice Documents and place them on the Advice Site. After that, these documents will be automatically diffused to all Advice Subscribers using standard Internet protocols. The process is Subscriber-driven – as Advice Readers on individual machines check-in to the site at random times, they obtain and read the new advice which has been published since the last visit.
- The Advice Subscriber does not need to do anything on a routine basis. He does not have to read advice documents or even be particularly able to understand technical issues associated with the computer. Instead, he simply ignores the existence of the AdviceNet system *until* AdviceNet *asks for attention.* At that point he reads the *advisory* being presented by the Advice Reader, which describes a potential problem and solution, and then 'clicks OK' to permit the solution.

The AdviceNet system directly addresses several 'hot-button' issues:

- The Advice Provider uses this system to minimize technical support costs. In particular, AdviceNet truly enables the TSO to deal with a problem just once; for after identifying a problem and solution, it can publish that information and automatically forestall future calls.
- The Advice Provider who uses the AdviceNet system wisely not need ~~to~~ worry about users becoming angry from receiving unsolicited information. Users are not asked to know about or review advice unless and until there is advice which is *Relevant* and *Timely* to the user.
- A user can be confident that the advice being shown to him was in fact authored by the source ~~they are~~ he is subscribing to (and presumably therefore trust). He is in this way protected from unsolicited advice of all kinds, in particular 'junk advice' or 'virus advice'.

3.3 AdviceNet Technology

The AdviceNet system probably sounds simple and efficient. In fact the simplicity derives from hard work! Universe Communications, Inc. has developed specialized technology in three different areas to make possible a simple and efficient solution.

- *Communications Protocols.* Currently, the Internet has three main branches: e-mail, World Wide Web, and Usenet. Viewed as communications protocols, these branches are distinguished by the type of communication – one-one, one-many, many-many – and the means of access. In essence, AdviceNet is a fourth branch of the Internet. It has similarities and essential differences from each of the other branches of the internet. The message format is similar to e-mail. The subscription mechanism is reminiscent of USENET. The site structure is reminiscent of WWW. Yet it is not identical to any of these. For example, Advice Providers do not 'mail' information to Advice Readers; and Advice Providers cannot in general expect that their advice will routinely be read by users; in fact it is expected that most advice on a user's machine will be irrelevant and will never be shown to the user.
- *Relevance Language and Engine.* At the center of the AdviceNet system is a formal language for determining the Relevance of messages to the current PC state. This language allows advice providers to specify quite precisely the situations under which a certain piece of advice will become relevant. The providers can use the language to specify that advice will be relevant only to certain manufacturer model numbers, only to certain versions of installed software, only under certain control panel settings, only under certain file attributes, and so on.
- *Inspectors.* The ability to describe the characteristics of a machine rests on the AdviceNet Inspector library, which allows the AdviceNet reader to inspect the machine for various attributes – versions and models of hardware and software, values of system settings, file attributes etc.

These three contributions represent a breakthrough in the conceptualization of support information. They provide a universal mechanism for automatically recognizing problematic situations and a universal mechanism for automatically providing solutions relevant just in those situations. For more information about the AdviceNet technology, see section 5.

3.4 How it works

In order to make the functioning of the AdviceNet system more concrete, we describe some typical applications. In order to make the reading more vivid, we take specific product names and true problem scenarios and describe how these problems would be solved if the organizations involved were using AdviceNet to solve their problems.

3.4.1 Aladdin and Netscape

Aladdin Systems is a maker of data compression products. It sells a premier data compression and archiving solution – Stuffit Deluxe 4.0 – to the Macintosh market. It also provides an OEM product – Aladdin Expander – that has a subset of the features of the deluxe product and which is installed by Netscape Navigator 3.

Here is a real problem situation. It turns out that when Netscape Navigator is installed *after* Stuffit Deluxe, the installer wipes out certain system settings in the process of setting up Expander. As a result, Stuffit no longer works properly – many of its functions become disabled. Also, the user no longer has available the ability to automatically invoke the functionality of Stuffit Deluxe

On the other hand, for users who installed Stuffit Deluxe *after* Netscape 3.0, the problem does not occur.

Here is a real problem that Aladdin has faced: some, but not all, users of their premier product no longer have working products. Let's say that at the time of this problem, 75% of the Aladdin user base employed Stuffit Deluxe. Let's assume that half of those users installed Netscape before, and half installed Netscape after, the Stuffit Deluxe Application. So about three-eighths of Aladdin's users are affected by this situation.

Under current support regimes, Aladdin has these options

- A mailing to all registered users, costing a great deal and reaching only a fraction of the user population.
- Waiting for 1-800 phone calls from three-eighths of Aladdin's users; costing a great deal in support staff time and in phone charges.

Neither option is really effective or really efficient

Suppose instead that Aladdin had been able to employ ADVICENET as part of its technical support strategy. Then it could rely on the fact that upon installation of Stuffit Deluxe on a user's machine, an Advice Reader was installed and was subscribing to the Aladdin Advice Site. So Aladdin could simply publish an advisory describing precisely the combination of circumstances under which the problem will arise, and providing a solution, in the form of a simple AppleScript file to change various system settings. Minutes after the advisory was published, Advice Readers worldwide would begin to obtain copies of the new advisory. Those users who were directly affected would be notified of the situation and would be asked if they would like to fix it; upon approval, the fix would be provided.

The cost to Aladdin of this solution: training and keeping a ~~programmer~~ tech services who understands how to author ADVICENET Advice Documents, and owning the basic Advice Provider tools. The savings, as compared to mailing or phone support? Highly significant.

tech
ccea

3.4.2 Global Village and Eudora

Global Village is a manufacturer of Modems and other hardware, primarily for the Macintosh marketplace. In 1996 GV released a PCMCIA card for Macintosh Powerbook laptops called PowerPort Platinum, which combined a modem and ethernet card on a single PCMCIA card.

The original GV driver had a problem when the Ethernet card was used in conjunction with certain TCP/IP operations, which showed up as a failure of Qualcomm's e-mail program Eudora to finish transfers of attachment files.

The problem therefore generated calls to Eudora's tech support organization (rather than GV's), and after a while, both Qualcomm and GV were aware of the problem. The solution was rather simple, involving an update of the GV driver, and was made available by the GV web site and the Eudora Web Site. However, neither organization attempted to notify users of the problem and solution. Many users only found out about the problem and solution by speaking with tech support. The result was that the problem was never really solved; it has kept generating calls to tech support even up to the present time.

Suppose now that GV and/or Qualcomm had employed ADVICENET as part of their technical support strategies. Then they could rely on the fact that upon installation of either the GV PC card driver or of Eudora, the Advice Reader was installed and was subscribing to the Advice Site of GV or Qualcomm. So GV could simply publish an advisory describing precisely the affected equipment – Apple Powerbooks with GV PowerPort Platinum 1.1.1 driver or earlier – and providing the solution, in the form of a simple script that would download and install a new driver. It could notify Qualcomm that this advisory was something that should be offered also to Eudora users, and Qualcomm would then in response, mirror the advisory at the Eudora Advice Site. Minutes after the advisory was published – by both Qualcomm and GV – Advice Readers worldwide would begin to obtain the new advisory, and PowerPort Platinum users everywhere would begin to be notified of the situation and would be asked if they would like to fix it; upon approval, the fix would be provided. Those users of Eudora – or of other GV products – who were not PowerPort Platinum customers would not be notified.

The cost to GV/Qualcomm of this solution: training a ~~programmer~~ who understands how to author ADVICENET Advice Documents, and acquiring the basic Advice Provider tools. The savings, as compared to servicing numerous user telephone inquiries? Highly significant.

3.4.3 Mindspring and PSI Net

Mindspring is an Internet Service Provider which has grown dramatically over the last year. In 1997, Mindspring received a poor rating in certain service listings owing to poor connection rates at certain Points-of-Presence (POPs). Mindspring identified the problem as localized mainly to those POPs that were owned by PSI, a nationwide networking company. Mindspring developed its own alternative to those POPs but was faced with the problem of migrating its user community from the PSI POPs to the new Mindspring POPs. Despite repeated e-mails to the user population, it turned out to be difficult to get users to change the telephone numbers in their PPP settings.

The situation forced Mindspring to pay for having two POPs in many local areas – the new Mindspring POP, accessed by energetic and dutiful customers who had read and followed the Mindspring recommendations – and the old PSI POP, accessed by lazier souls. The solution which users were being asked to implement on their own was rather simple, involving simply typing in a new phone number in a certain control panel, and the approach was spelled out in the e-mail messages to all users. But the rate of user cooperation was well below 100%.

Suppose now that Mindspring had employed ADVICENET as part of its technical support strategy. Then it could rely on the fact that upon installation of the Mindspring service software, the Advice Reader was installed and was subscribing to the Advice Site of Mindspring. So Mindspring could simply publish an advisory describing precisely the combination of circumstances under which the problem would arise, *isolating exactly those user computers referencing a PSI POP* and providing a solution, in the form of a simple script to change the system settings as required. Minutes after the advisory was published by Mindspring, Advice Readers worldwide would begin to obtain the new document, and those users who were directly affected would be notified of the situation and would be asked if they would like to fix it; upon approval, the fix would be provided. Those Mindspring users who were not invoking PSI POPs would not be notified.

The cost to Mindspring of this solution: training and keeping a ~~programmer~~ who understands how to author ADVICENET Advice Documents, and owning the basic Advice Provider tools. The savings, as compared to continuing to pay PSI for use of PSI POPs?

Highly significant.

3.4.4 Novell and the Year 2000.

Novell is a large manufacturer of networking and network management software. In November of 1997, it was reported that Netware 3.12 had a year 2000 bug: after the rollover of the real date from 1999 to 2000, the Novell date would begin to show 1988. This turns out to be a complex problem to solve because a fraction of Novell's user base uses Netware 3.12; there is currently no convenient tool to identify and patch Netware 3.12 clients which are present in a large heterogeneous network including machines speaking other revisions of Netware and other protocols enirely.

Suppose now that Novell had employed ADVICENET as part of its technical support strategy. Then they could rely on the fact that upon installation of the original Netware software, the Advice Reader was installed and was subscribing to the Advice Site of Novell. So Novell could simply publish an advisory describing precisely the combination of circumstances under which the problem would arise, *isolating exactly those PC's using Netware 3.12* and providing a solution, in the form of a simple patch and installer scripts to change the software as required. Minutes after the advisory was published by Novell, those users who were directly affected would be notified of the situation and would be asked if they would like to fix it; upon approval, the fix would be provided. Those Netware users who were not running Netware 3.12 would not be notified.

The cost to Novell of this solution: training and keeping a programmer who understands how to author ADVICENET Advice Documents, and owning the basic Advice Provider tools. The savings for Novell as compared to answering many phone calls or using valuable account representative time exclusively for this purpose: Highly significant.

A variant of this would be available to very large organizations running Netware as part of their intranet. They would create their own advice site and set up their own PCs to subscribe to their corporate advice site. They could then mirror on the corporate site just those items from the Novell site which match corporate policy. Doing this, they would get all the benefit of Novell expertise, while controlling the impact of Novell-authored advice on their own assets.

The cost to the Corporate IT department of this solution: training and keeping a ~~programmer~~ who understands how to author ADVICENET Advice Documents, and owning the basic Advice Provider tools. The savings for Corporate IT as compared to identifying affected machines and manually upgrading each machine: Highly significant.

3.5 Compatibility with Knowledge-Base Tools

A key fact about ADVICENET is that it is complementary to existing automated technical support technologies. Rather than competing with existing knowledge-bases, it works with them, leveraging the investment in existing knowledge bases.

For example, large software companies like Microsoft are currently developing very large techincal knowledge bases, holding information about thousands of potential problems and methods of solution. Microsoft recently spent $500 million on developing its current technical support library and web site [7]. However, such a large investment in such resources has a limited payoff. The information must sit passively in server databases, until a tech support professional or a sophisticated user taps into the information to find the precise information of relevance to them.

AdviceNet allows TSO's, over time, to convert the information in their knowledge bases from passive to active information – mobile information which is in effect seeking out user PC's and pro-actively solving various problems on those PC's.

We advocate a 'Publish as you Go' approach to such conversion. Under this conversion scheme, whenever a support technician identifies and solves a user call-in problem using the knowledge base, the technician immediately publishes an advice document based on the given knowledge base document. (Authoring such documents is rather straightforward). The knowledge from the knowledge base is then in this way put to work actively avoiding future calls on the same issue.

'Publish as you Go' is much more efficient than 'Convert Everything'. Rather than spending a lot of time publishing advice to prevent problems which are largely theoretical, the TSO is preventing the problems that actually are happening.

3.6 Compatibility with Relationship Management Tools

Once the TSO has adopted AdviceNet as part of its support process, it will obtain many side benefits. The organization will have in-house ~~an~~ individuals with experience authoring AdviceNet documents, and the organization will be able to author documents which help in other phases of the customer relationship. We give a few examples:

- *Before the Sale.* The AdviceNet system can be used to automatically produce *Compatibility Checkers* from the TSO's advice database. These help users to verify that an intended purchase of software or hardware is compatible with an existing user system. This helps the TSO avoid costly expenses from tech support calls due to product incompatibilities or from outright returns.

  In order to gain this benefit the TSO has only to publish an advice digest at its web site which compiles all existing advice on compatibility issues in one large advice digest. The prospective user can download and presents the digest to the Advice Reader in order to see what advisories would be generated upon installation of the new product.

- *Upon Installation.* The AdviceNet system may be used to automatically produce an *Initial System Profile* of a registered user. The profiler document surveys the user's configuration at registration time, and automatically (upon user approval) returns the information to the TSO. This allows the TSO's database to maintain a record describing the user configuration, which will lead to shorter tech support calls when they occur. Because AdviceNet in effect fills out the registration form automatically – at least as far as machine configuration information goes – the process is less burdensome on users and more informative to providers. Because users see the future use of AdviceNet as a benefit to them, they are more likely to cooperate with the registration process.

  In order to gain this additional benefit the TSO has only to include, as part of the installation process of its software product, a special advice document which upon installation queries the user's system to automatically learn about the user's configuration and then offer to return that information to the TSO. The user's privacy and security are respected; the user can see what information is being provided and only upon user aproval will the information be relayed back to the TSO.

- *Tech Support Event.* The AdviceNet system may be used to automatically produce an *Event-Driven Profile* in order to speed up technical support calls. This advice document in effect surveys the user's configuration It automatically (upon user approval) returns the information to the TSO, and even, if desired, arranges for a technical support telephone call appointment. This allows the TSO to update its database records describing the user configuration, which will lead to a shorter tech support call, and allows the user, if desired, to schedule a call from tech support and so avoid waiting in line for a long time on a 1-800 phone line.

In order to gain this additional benefit the TSO has only to include, as part of the installation process of its software product, an advice document which is activated upon user request, queries the system to automatically learn information about the user's configuration and then offers to return that information to the TSO and to set up a telephone appointment. The user's privacy and security are again respected.

- *Upgrade/Update event.* The ADVICENET system may be used to announce the existence of updates and upgrades to a widespread user population. This allows the TSO to minimize the size of mailings which are customary today.

  In order to gain this additional benefit the TSO has only to author an advice document which describes the type of computer configuration for which the update is intended and and which links to the appropriate file. Then it simply publishes the document at the Advice site.

In short, the cultivation of an in-house advice authoring capability can pay off in many additional benefits beyond technical support: increased user satisfaction, increased participation in registration and updates, shortened tech support calls.

4 The Future of Technical Support

As ADVICENET becomes integrated into the technical support industry, the world of technical support will change, both for users and providers. We'll give here a detailed sketch of how this may go.

4.1 The Independent User

~~Under the ADVICENET paradigm,~~ the small ~~business user~~ will typically have installed on his computer a range of peripherals and of software applications. He will automatically be a subscriber to the advice sites of the manufacturers of these products. He will typically also be a subscriber to the advice sites of certain user groups and perhaps also to advice sites of certain systems integration consultants.

Of course, the advice sites of hardware and software manufacturers will contain announcements of updates to drivers and to installed software, and will also contain advisories about problems which could be experienced by users who have certain hardware and/or software applications installed, or who have certain system settings in their configuration files. The advice sites of systems integration consultants may contain custom advice, authored by the consultant, to better run the small business computer clients of his consulting practice. The advice sites of the user groups may contain diverse information – one can imagine information about interesting enhancements to major applications of interest to such users or about useful shareware.

In short, the typical consumer will have many one-one relationships with many different organizations, all managed and streamlined through the ADVICENET reader.

4.2 Relationship Management

AdviceNet provides a centralized, organized, unified and efficient way to mediate communications between providers and consumers.

Consumers will come to recognize many benefits of the AdviceNet system.

- *Quality.* The user will receive better service, will experience fewer problems, and will spend less of his own time in solving these problems. The user will understand the AdviceNet approach and will prefer it to traditional mechanisms because of its superior effectiveness and speed.
- *Simplicity.* There will be one well-understood user interface for communications of all kinds with providers of all kinds.
- *Ease.* There will be no need for the user or for the provider to do anything 'by hand' in order for the user to receive notices appropriate to his situation.
- *Freedom from Worry.* The user will have his security and privacy protected, and will understand the philosophy and the technology by which this happens.
- *Protection from Abuse.* The user will be protected from nuisance messages by AdviceNet , and will understand the philosophy and the technology by which this happens. The user will have natural, widely understood mechanisms to correct abusive situations.

In some sense, AdviceNet will become a way that the user can concretely visualize the technical support industry: it will be the user interface to that industry, and will teach users what to expect and what to obtain from tech support.

In turn, this will create new commercial opportunities as new businesses are formed around the use of this well-understood communications tool.

4.3 Support Industry Benefits

The support industry is in its infancy. While ideas like knowledge bases and trouble tickets are beginning to be standardized [6], there is in general no accepted set of tools for dealing with technical support problems in the computer industry.

AdviceNet changes that. It makes concrete these four major principles:

- Each Problem Gets Addressed Once and for All
- Problems get addressed pro-actively
- Problems get diagnosed automatically
- Problems get solved automatically

The AdviceNet system turns lofty ideals into practically acheivable daily operating facts. AdviceNet comes not a moment too soon.

Adhering to these principles is essential for the computer business to continue its current growth.

The usual approaches for making computers work well have not been designed to cope with the explosively growing size of the number of computers and users, or with the increasing number of potentially problematic interactions. In fact, continuing current trends, the day is coming when there will not be enough 1-800 telephone staff, nor enough dealers and systems consultants, to deal even minimally with all the support problems that will arise.

However, ADVICENET goes beyond these four principles, and establishes new possibilities that had not been considered before

- *Precise field upgrades.* A small subset of a widely distributed user base can be updated or upgraded easily.
- *Precise problem feedback.* A two way channel can be set up, relaying information about the existence of unforeseen, serious problems back to the provider.
- *Increased registration rate with user base.* The provider can make it easier and more attractive to users to register, by automating the process and by cheaply provide benefits to users which will maintain contact.
- *Increased communications with user base.* The provider can stay in better contact with the user base, and canincrease the upgrade rate.

4.4 The Corporate User

A variety of possibilities exist for the use of ADVICENET in a corporate environment which we have not so far discussed. The corporate user will typically have installed on his computer a range of peripherals and of software applications. Depending on the corporate policy, there are two extreme cases. On the one hand, the computer might be configured much as an independent user's computer would be, with a range of subscriptions outside the intranet, some of them recommended by the corporate policy, others simply improvised by the user himself. At the other extreme, the computer might be permitted to subscribe only to advice at the corporate advice site. In the second scenario, the corporate site would be maintained by the IT department, which would review the advice being published by the producers of the software which had been adopted by the company, and would selectively re-publish advisories from those external organizations. In the second scenario – that of a rigorously controlled intranet environment – ADVICENET might typically be configured in a more 'automatic mode' and could be used to automatically run relevant advisories without user intervention, and to automatically report back information about corporate assets to the IT department without user intervention.

5 To Probe Further

This document give~~s only~~ e a broad overview of the ADVICENET system, focusing on the aspects which would be important to TSO's. There are several other documents published by Universal Communications, Inc. which should be consulted for more detailed information about the systems described here. Four of these documents could be read immediately after this document.

- THE ADVICENET SYSTEM. Describes the ADVICENET system for communicating active advisories to computers worldwide. Describes in functional terms the key components and protocols and points to other documents which contain further information.

- The AdviceNet Site Developer's Manual. Describes the components of an advice site and how an advice provider can construct those components. Illustrates how an advice site can save an advice provider money in technical support and maintenance costs.
- The AdviceNet Relevance Language. Describes the Relevance language which allows the Advice provider to describe the machines to which a certain pe-iec of advice will be relevant.
- The AdviceNet Inspector API. Describes how an advice provider can extend the Relevance language to include capabilities directly addressing specific needs of the advice provider. Gives detailed information on the Applications Programming Interface and on the programming environment which is required to develop Inspectors.

This, and a list of all documents published by AdviceNet , can be obtained directly from Universe Communications, Inc.

References

[1] The AdviceNet System.

[2] The AdviceNet Site Developer's Manual.

[3] The AdviceNet Inspector API.

[4] Comerford, Richard (1996) The battle for the desktop. *IEEE Spectrum* May 1997. pp. 21-

[5] 1997 Technical Support Cost Ratios Survey. Published by Association of Support Professionals.

[6] Law, Bruce (1996) A New Support Standards Initiative. Published by Association of Support Professionals. `http://www.asp.org`

[7] Tarter, Jeffrey (1996). Is there a payoff for service quality? Published by Association of Support Professionals. `http://www.asp.org`

List of Prior [illegible]

010[illegible]

EXHIBIT F

SCOTT - DOC [illegible]

Outline for Meeting with Michael Glenn

MEDICAL ADVISOR [illegible]

[illegible]

THE [illegible]

1. Problem being addressed

Advice Provider:

Organization or individual represented by server on an intranet or internet.
Knows of conditions under which certain consumers would like to know something,
    potentially to act on it with approval.
Potentially thousands or millions of conditions it can offer advice about.
Potentially millions or billions of individuals it can offer advice to
Most conditions depend on very special combination of circumstances at consumer end
    Affect only a small fraction of consumer base, but large number of consumers
    Description of condition might involve knowing a great deal of detailed information
        about the computer or contents of its storage devices
This information might be considered very sensitive by consumers

Advice Consumer:

Organization or individual on an intranet or extranet.
Knows of Advice Providers offering advice of potential benefit to consumer.
Typically does not want to review all the advice being offered by Advice Provider; only
    wants to see the subset of advice which is relevant and timely.
Typically does not want to reveal information about his identity or detailed condition of his
    computer or contents of its storage devices to Advice Provider.
Wants possibility of evaluating advice before adopting it.
Wants possibility of adopting advice automatically.
Can have relationship of this type with tens or thousands of Advice Providers.

AdviceNet System:

Connecting AP with AC
    automatically matching consumers with relevant advisories
Preserving security, privacy,
    no consumer need reveal identity nor attributes
        no extra risk to consumer if advice not followed
Offering relevance, timeliness, activity
    advice typically only visible if relevant to consumer's situation
    relevance determined automatically by AdviceNet system
    relevance can include complex combination of
        timeliness,
        hardware attributes,
        database attributes
        personal attributes
        randomization
        remote attributes

- DEMO
- DISCUSSION OF TECH
- DISCUSSION OF SYSTEM

ABOUT QUESTIONS [illegible]
12 PAGES

Concrete Instance: Technical Support Industry

AP offers hardware, software, internet service, IT service
AP knows of problematic situations
AP knows precise description of situation preconditions
AP knows precise solution
AP packages information as advisory
AP offers by internet, using Advice Server
AC subscribes
AC has Advice Reader application
Advice Reader reads advisories, determines relevance
Reader reviews, approves, denies
On approval, automatic solution download/install/execute

There are many other concrete instances.

2. AdviceNet System Components

AP --	Advice Site	Internet URL-addressible directory
		plurality of files (advisories)
		directory message server
		http/ftp server
	advisories	typically specify precondition in formal language
		typically explain precondition in human language
		typically explain solution in human language
	Potentially	~~typically~~ offers internet access to solution to problem
		specially formatted ~~ascii~~ file
		easily transported over internet
		easily created/maintained
	site description file	describes an advice site
		basis for initiating subscription by consumer
		location (URL)
		frequency of synchronization
		type of relationship (free/fee)
	inspector libraries	Special purpose executable code
		Extends relevance language
		advice-site specific extension
AC --	advice reader	Application running on client machine
		Synchronizes with Advice Site
		Fetches advice files
		Unwraps advice messages
		Stores advice messages locally
		Interprets Relevance
		Displays Relevant messages to user
		Manages bodies of advice
		Manages subscriptions
	inspectors	Invoked by Advice Reader
		Can inspect properties of machine environment
		Inspect hardware, file system properties, data in files
		Allow Relevance decision to involve complex environmental properties.
	user profile	Special user-created file
		user-input data on preferences, requirements
	~~Wizards~~	support solution process
		Mr. FixIt
		Mr. Shell
	advice digests	
	advice pools	
	site profiles	

3. Transaction overview

Subscription model	Users become aware of existence of advice sites Users subscribe
Site Synchronization	Periodically or under user control Advice Reader queries advice site for directory message If directory changed, synchronizes contents Downloads advice from server to client Deletes moot advice on client
Downloading	Uses FTP/HTTP internet mechanism
Interpretation	Advisory: potentially complex hierarchical structure Reader unpacks the components of structure
Relevance Evaluation	Uses formal Relevance Language Parses Relevance Clauses Evaluates Clauses

Inspectors	Evaluate certain phrases in language
	Called by Advice Reader @ Relevance Evaluation
	Obtains system information
	    File Properties
	    System Settings
	    File contents
	    Hardware Properties
	    User profile
	    Remote File/System Properties
	    Randomness

Digital Authentication	digital signature on advisory digital authentication of advisory contents
Display	List of relevant advisories Organizational and management tools
Action	User Offline action Download/install/execute Solution Script File for Wizard

4. Philosophical Overview

Automatic Unattended Operation
- infrequent user involvement
- periodic unattended synchronizations downloads
- constant unattended relevance evaluations

Privacy
- One-way nature of subscription interaction
- Customer need not reveal information -- name or preferences
- Information on client machine stays on the machine

Security
- Typically, no advice without subscription
- Subscription connotes partial trust
- Typically, no effects on system without prior notification and approval
- Security through trust
- Internet infrastructure allows retraction of one's own faulty advice
- Internet infrastructure allows criticism of other people's faulty advice

Decentralization
- any IP can be server -- no central registration
- any IP can be client -- no special registration

Extensibility
- anyone can extend the language to give it new capabilities

5. Internet Meta-Functions

Advice Reader typically subscribes to three privileged sites

`Advisories.com`
- Distribution of Advice Reader
- Distribution of Subscription Information

`BetterAdviceBureau.org`
- Issue advisories against bad advisories
- Issue warnings against bad advice sites
- Compile advice site complaints

`UrgentAdvice.net`
- Issue urgent advice

6. Advice File Format

Purpose	to package one or several advisories to offer one or several variants of same advice.
Basic Message:	Wrapper Subject Line Relevance Clause Message Body Action Button
Hierarchical	Advice file = digest of one or more messages Each component potentially relevance-guarded
MIME	Implementation of Wrapper Uses internet standards to package group of messages RFC 822 and successors
Message Body	HTML text both text and HTML
Authentication	Each component of message can be signed Each component of message can be authenticated Digital authentication -- MD5 or similar Digital signature -- PGP or similar

7. Relevance Language

Purpose	To specify precisely conditions under which a certain message would be relevant Ability to describe system state, including hardware, files, file contents, network and remote system states Ability to refer someday to objects in system and world not yet known/created
Characteristics	Descriptive Language English-like Object-Oriented, Strongly Typed Not a procedural language
Limitations	Despite visual resemblance, not like programming language No "`If`" "`While`" "`Case`" "`Goto`" "`Switch`" No traditional variables in language Computed arguments not allowed
Security	Evaluation must halt -- No infinite loops No ability to change -- no effects to file system
Extensibility Mechanism	Language can be expanded by adding inspector libraries Defines new data structures Defines new behaviors of those data structures

8. Inspector Libraries

Object-oriented structure	Specify object type Specify all allowed properties Specify outcome type of all allowed properties
Base library	Platform Specific
Extension libraries	Add new components dynamically
Functional Component	Need for extension Need for security
Intellectual Component	Soul of Machine

Special Inspectors

Registry Inspector	find any property of registry
Database Inspector	find any property of general SQL database
User Profile Inspector	find any property of user's profile
Remote Inspector	find any property of other machine in special trust relationship

9. Variations

Situational Advice

Alternate Transport mechanism
- Advice by e-mail
- Advice by drag and drop
- Advice by file reference

Open Bi-directional communications

Questionnaires	e-mail back inspector values

Masked Bi-directional communications
- Anonymity
- Randomization

Charge Money (metering & cyber$$)
- to publish advice
- to subscribe
- to synchronize
- to download advice
- to download inspector
- to download solution
- to invoke solution

10. Spinoff Opportunities

Commodity Markets
Safety Warnings
Operational Advisories
Product recalls
Portfolio Management
[illegible]

11. Important System Components worth protection

[illegible] Overall System
Advice Site
Better Advice Bureau Site
Urgent Advice Site
Advice Reader
Inspector libraries
Relevance Language
Document types
Optimizations
Registry Inspectors
Database Inspectors
Questionnaires
Randomized response
Remote Relevance Invocation
Wizards: Mr Shell, Mr. Fixit
Automatic advice generation: Naildown

12. Claims

About Relevance-Guarded Messaging

Ability for authors of broadcast messages to specify precisely the conditions under which a message would be of interest to consumer, by setting detailed conditions known only within user environment. Does not require breach of consumer privacy/security

About AdviceNet System

Automatically connect a population of consumers to *relevant* advisories.
Automatically furnish solutions to those consumers having relevant advisories.
Relevance based on detailed knowledge of information and resources at consumer's site or proxies.
System maintains privacy relationship: although it inspects sensitive information on consumer's site, the advice provider learns nothing about that site through the inspection process .

About Priveleged Advice Sites

`Advisories.com, BetterAdviceBureau.com, UrgentAdvice.com`
Mechanism to maintain integrity of an advisory system
Mechanism to identify released advisories and deprecate them

About Inspectors

Mechanism for authors to write in language which can remotely inspect properties of hardware, files, system settings, user profiles, file contents, database contents

Mechanism for authors to extend such language through provision of libraries dynamically extending language syntax, data structures, and semantics

Applications outside of AdviceNet System

About Registry/Database Inspector

Extension of AdviceNet mechanism to know properties recorded in any Microsoft Windows '95/'98 database
Extension of AdviceNet mechanism to know properties recorded in any standard SQL database.
Applications outside of AdviceNet System

About Remote Inspectors

Extension of AdviceNet mechanism to obtain information about user's environment to data repositories that may lie outside the user's jurisdiction/control. Allows to maintain privacy of distributed sensitive data.

Applications outside of AdviceNet System

About Questionnaires

Mechanism for Data Seeker to enlist consumer populations with special characteristics in data gathering, obtaining automatically detailed information about consumer's situation while offering full consumer control of what is being communicated back to Data Seeker.

About Randomized Response

Mechanism for Data Seeker to enlist consumer populations with special characteristics in data gathering, obtaining automatically detailed information about consumer's situation while actually obtaining no information about any individual consumer.

Remaining Questions:

Wizards
Optimizations
Commodities Markets
Financial Management

12. Claims

About Relevance-Guarded Messaging

Ability for authors of broadcast messages to specify precisely the conditions under which a message would be of interest to consumer, by setting detailed conditions known only within user environment. Does not require breach of consumer privacy/security

About AdviceNet System

Automatically connect a population of consumers to *relevant* advisories.
Automatically furnish solutions to those consumers having relevant advisories.
Relevance based on detailed knowledge of information and resources at consumer's site or proxies.
System maintains privacy relationship: although it inspects sensitive information on consumer's site, the advice provider learns nothing about that site through the inspection process

About Priveleged Advice Sites

`Advisories.com, BetterAdviceBureau.com, UrgentAdvice.com`
Mechanism to maintain integrity of an advisory system
Mechanism to identify released advisories and deprecate them

About Inspectors

Mechanism for authors to write in language which can remotely inspect properties of hardware, files, system settings, user profiles, file contents, database contents

Mechanism for authors to extend such language through provision of libraries dynamically extending language syntax, data structures, and semantics

Applications outside of AdviceNet System

About Registry/Database Inspector

Extension of AdviceNet mechanism to know properties recorded in any Microsoft Windows '95/'98 database
Extension of AdviceNet mechanism to know properties recorded in any standard SQL database.
Applications outside of AdviceNet System

About Remote Inspectors

Extension of AdviceNet mechanism to obtain information about user's environment to data repositories that may lie outside the user's jurisdiction/control. Allows to maintain privacy of distributed sensitive data.

Applications outside of AdviceNet System

About Questionnaires

Mechanism for Data Seeker to enlist consumer populations with special characteristics in data gathering, obtaining automatically detailed information about consumer's situation while offering full consumer control of what is being communicated back to Data Seeker.

About Randomized Response

Mechanism for Data Seeker to enlist consumer populations with special characteristics in data gathering, obtaining automatically detailed information about consumer's situation while actually obtaining no information about any individual consumer.

Remaining Questions:
- Wizards
- Optimizations
- Commodities Markets
- Financial Management